

SRT Guida (pratica) completa

A cura di Danilo Belvedere, Andrea Bisacchi, Davide Giordano e Fabrizio Torriano

PROBLEMA 1

NB: Una qualsiasi risposta conclusiva di tipo S/S significa che i TEST successivi non sono da fare.

In questo formulario avrete in **ROSSO** tutte le formule da applicare / calcoli da effettuare.

Troverete invece in **VIOLA** i risultati da inserire all'interno del testo che verrà consegnato.

NB: Tutti i test sono conclusivi quindi le risposte da dare sono **Sì/Sì** oppure **No/No** tranne per AUDSLEY (che può avere N/S oppure S/S) e DMPO dove segnato nella guida.

PREREQUISITI:

Vengono forniti dal testo un certo numero di processi P_i sottoforma di Tabella:

	T_i [t.u.]	C_i [t.u.]
P_1	10	2
P_2	15	4
P_3	20	5
P_4	30	3
P_5	60	2

- Inserire in tutti i casi la colonna **Ui (usata nel test di Han per esempio)** affianco alla colonna dei **Ci** calcolata come **Ci/Ti**.

- **Di** viene fornito dal testo, se non specificato **Di = Ti**.

Nell'esempio affianco $U_1 = 2/10 = 0.2$

- Calcolare **Up** come la **somma di tutti gli Ui** calcolati precedentemente.

- Aggiungere la colonna **di**, ogni valore è uguale a **Di/Ti**

a) Dimensionamento del ciclo MAGGIORE:

M = mcm (Ti) = mcm (10,15,20,30,60) = 60.

Per non perdere tempo, MCM online

(<https://www.okpedia.it/tool/calcolo-minimo-comune-multiplo-online>).

b) Dimensionamento del ciclo MINORE:

In questo punto bisogna restringere il numero possibile di m ad un insieme finito.

L'insieme di partenza da considerare è il seguente

max(Ci) <= m <= min (Ti)

Nell'esempio avremo quindi **5 <= m <= 10** ovvero **m = {5,6,7,8,9,10}**

Restringere ulteriormente l'insieme degli m appena trovati applicando

M mod m = 0

Tentare con tutti i numeri trovati nell'insieme precedente ed escludere i valori che non rispettano tale condizione. (Resto della divisione uguale a 0)

ES: **60 mod 5 OK** mentre **60 mod 7 NO**

m = {5,6,10}

L'ultima condizione da applicare per restringere m richiede il calcolo del MCD.

La formula da applicare per verificare se ognuno degli m trovati rispetta le condizioni è la seguente

$$2*m - \text{MCD}(m, T_i) \leq T_i \quad \forall i$$

In sintesi si effettua il MCD tra m considerato e **TUTTI i T_i** verificando che tale valore sia inferiore o uguale al T_i stesso considerato.

In caso di parità/più m disponibili, si sceglie quello più elevato.

CONSIGLIO: Partire da m più elevato possibile perchè al primo che becchi concorde con la formula puoi fermarti! (non troverai mai m migliori di esso).

Per velocizzare il calcolo del MCD usare:

(<https://www.okpedia.it/tool/calcolo-massimo-comune-divisore-online#tool>)

Nel caso in cui si voglia calcolare manualmente occorre fare la scomposizione in fattori primi dei due numeri presi in esame e, dati i due insiemi della scomposizione, prendere come RISULTATO MCD il **prodotto** tra i valori **comuni** ai due insiemi con l'esponente più piccolo.

Es: $10 = 2*5$ e $20 = 2^2 * 5$ quindi $\text{MCD}(10, 20) = 2*5 = 10$.

Es: $m = 10$ $2*m - \text{MCD}(m, T_i) \leq T_i$

Alla prima iterazione di esempio avremo

$$2*m - \text{MCD}(10, T_i) \leq 10$$

Calcolare con il TOOL l'MCD tra il valore corrente m e l'intera colonna dei T_i una alla volta:

MCD tra: (10,10),(10,15),(10,20),(10,30),(10,60) e verificare se l'insieme dei valori risulta sempre \leq del rispettivo T_i .

$$20 - \text{MCD}(10, 10) \leq 10 \quad 20 - 10 \leq 10 \text{ SI}$$

$$20 - \text{MCD}(10, 15) \leq 15 \quad 20 - 5 \leq 15 \text{ SI}$$

$$20 - \text{MCD}(10, 20) \leq 20 \quad 20 - 10 \leq 20 \text{ SI}$$

$$20 - \text{MCD}(10, 30) \leq 30 \quad 20 - 10 \leq 30 \text{ SI}$$

$$20 - \text{MCD}(10, 60) \leq 60 \quad 20 - 10 \leq 60 \text{ SI}$$

Avendo trovato come concorde alla formula il valore di m più elevato possiamo fermarci, dunque:

$$m = z$$

c) Calcolo del numero dei cicli minori e dei Job per processo nell'ambito di ciascun ciclo maggiore:

$$\underline{ncm} = M / m = 60/10 = 6$$

$$nj1 = M / T1 = 60/10 = 6$$

$$nj2 = M / T2 = 60/15 = 4$$

$$nj3 = M / T3 = 60/20 = 5$$

$$nj4 = M / T4 = 60/30 = 3$$

$$nj5 = M / T5 = 60/60 = 1$$

d) Identificazione del ciclo minore(o dei cicli minori) in cui ciascun Job può' essere eseguito:

Creare la seguente tabella con

Numero righe = ncm (calcolato al punto c)

Numero colonne = nj1+nj2+nj3+nj4+nj5.

Esempio:

Dato che nj1 = 6 Avremo J11,J12,J13,J14,J15,J16

Dato che nj2 = 4 Avremo J21,J22,J23,J24 etc etc...

ES : Se M = 60 e J2x possiede 4 colonne avremo 4 intervalli distanti 15 l'uno dall'altro perche' $15 \cdot 4 = 60$.

M = 60

		0-10	10-20	20-30	30-40	40-50	50-60	0-15	15-30	30-45	45-60	0-20	20-40	40-60	0-30	30-60	0-60
		J ₁₁	J ₁₂	J ₁₃	J ₁₄	J ₁₅	J ₁₆	J ₂₁	J ₂₂	J ₂₃	J ₂₄	J ₃₁	J ₃₂	J ₃₃	J ₄₁	J ₄₂	J ₅₁
0-10	c ₁	X						X				X			X		X
10-20	c ₂		X									X			X		X
20-30	c ₃			X					X				X		X		X
30-40	c ₄				X					X			X			X	X
40-50	c ₅					X								X		X	X
50-60	c ₆						X				X			X		X	X

Crocettare l'intersezione nella tabella solamente se l'insieme della RIGA considerata rappresentante i Ci è INTERAMENTE CONTENUTA nell'insieme della COLONNA.

ES:

1)c₁ ha come intervallo 0-10 mentre J₁₁ ha come intervallo 0-10.

Essendo 0-10 interamente contenuto in 0-10 allora crocetto.

2)c₃ ha come intervallo 20-30 mentre J₂₁ ha come intervallo 0-15.

Essendo 20-30 NON interamente contenuto in 0-15 allora NON crocetto. Crocetterò per esempio J₂₂ in quanto 20-30 è interamente contenuto in 15-30.

e) Pianificazione dell'esecuzione di ciascun Job

La tabella appena fatta consente di produrre la tabella rappresentativa dell'esecuzione di ogni JOB.

Considerare che:

-Il numero di slot disponibili nella tabella (il numero di caselle bianche in cui posso piazzare un Job per ogni riga) è pari ad m.

-Ogni $J_{x,x}$ deve occupare nella tabella tanti posti quanti ne ha il suo rispettivo c_x del testo del problema iniziale.

Nel caso dell'esempio avremo che il posto occupato nella tabella da:

$J_{1,x}$ sarà di 2 blocchi ($C_1 = 2$)

$J_{2,x}$ sarà di 4 blocchi ($C_2 = 4$)

$J_{3,x}$ sarà di 5 blocchi ($C_3 = 5$)

$J_{4,x}$ sarà di 3 blocchi ($C_4 = 3$)

$J_{5,x}$ sarà di 2 blocchi ($C_5 = 2$)

Per inserire un Job nella tabella attuale, occorre **controllare la tabella del punto "d"** e **verificare dove sono state inserite le crocette**.

Esempio: **J11** sarà inseribile nella tabella SOLO nella riga **C1** mentre **J31** potrà essere inserito sia in **C1** che in **C2**. Analogamente **J41** potrà essere piazzato in **C1, C2** oppure **C3**.

Partire con l'inserimento dai **JOB più prioritari, ovvero i Job J1,X. Passare poi ai J2,X e così via**.

CONSIGLIO: Per evitare di dimenticarsi di posizionare in tabella uno qualsiasi dei Job, segnarsi a lato la LISTA dei Job da schedare nel seguente modo

J11, J12, J13, J14, J15, J16

J21, J22, J23, J24

J31, J32, J33, J34

J41, J42

J51

Ogni volta che si riesce a piazzare un Job CANCELLARLO dalla lista in modo da far rimanere solamente i Job non piazzati (se presenti) al termine dell'esercizio.

Preferire il riempimento di una riga intera piuttosto che lasciare spazi bianchi a caso.

Nell'esempio era meglio mettere J41 in C2 in quanto si riempiva completamente lo spazio per quella riga! [CONFERMATO OGGI DAL PROF]

m = 10 quindi 10 celle bianche per riga

J1,x = 2 blocchi

J2,x = 4 blocchi

J3,x = 5 blocchi

J4,x = 3 blocchi

J5,x = 2 blocchi

c1	J11		J21				J41				
c2	J12			J31			J51				
c3	J13		J22								
c4	J14		J23				J42				
c5	J15			J33							
c6	J16		J24								
		1	2	3	4	5	6	7	8	9	10

Dopo aver definito la tabella ci sarà da rispondere se l'esito è stato positivo per **tutti i Job**. In questo caso la risposta dipende dall'eventuale presenza di Job non inseriti (a causa della **mancanza di spazio**) nella tabella di sopra.

J32 per esempio e i suoi 5 spazi, sarebbe dovuto essere inserito il C3 oppure in C4 ma purtroppo lo spazio non era sufficiente.

In tal caso **si crocetta NO** e si indica **J32** come Job non inserito.

sì	no X	quali job ? J ₃₂
----	---------	--------------------------------

f) In caso di esito negativo, effettuare il rilassamento del minor numero possibile di vincoli

In questa sezione scrivere **Px (Ti,Ci)** del processo che non è stato schedato e **spezzarlo** come ci pare, in maniera da farlo entrare nella tabella.

ATTENZIONE. TUTTO P3 VERRÀ SPEZZATO QUINDI AVREMO DUE NUOVI JOB DA SCHEDULARE ovvero J32' e J32''.

Puoi scegliere tu in che modo spezzare (qui si spezza un $T_i = 5$ in due T_i pari a 3 e il secondo pari a 2).

Nel caso dell'esempio avremo quindi **J32 che fa parte di P3**, quindi:

P3 (5,20) => P3' (3,20), P3''(2,30) con $P3' < P3''$

Rifare la tabella considerando l'esistenza di J32' e J32'' di occupazione blocchi pari a:

J32' --> 3 blocchi al posto dei precedenti 5

J32'' --> 2 blocchi al posto dei precedenti 5

NB: cercare di riempire le celle il più possibile. Se puoi riempire una riga, fallo! Per esempio qui J41 può tranquillamente andare a C2 così da riempire la riga, idem J51 a C4.

m = 10 quindi 10 celle bianche per riga

J1.x = 2 blocchi	c1	J11		J21			J41					
J2.x = 4 blocchi	c2	J12		J31			J51					
J3.x = 5 blocchi	c3	J13		J22			J32'					
J32' = 3 blocchi	c4	J14		J23		J32''						
J32'' = 2 blocchi	c5	J15		J33								
J4.x = 3 blocchi	c6	J16		J24			J42					
J5.x = 2 blocchi			1	2	3	4	5	6	7	8	9	10

NB2: si può anche dividere in modo tale che 3 job da 4 diventino 4 job da 3

Test di Han

	T_i [t.u.]	C_i [t.u.]
P_1	10	2
P_2	15	4
P_3	20	5
P_4	30	3
P_5	60	2

Occorre definire l'insieme accelerato dei processi. Partire dai dati forniti dal problema relativi ai T_i e ai C_i di partenza.

L'obiettivo è trovare un insieme di processi la cui **sommatoria degli U_i sia ≤ 1** .

Per far passare il test di Han occorre infatti calcolare prima tutti gli U_i come $\rightarrow U_i = C_i / T_i$ Il test di Han è positivo se

$$Up = \sum_{i=0}^n \frac{C_i}{T_i} \leq 1.$$

FARE QUESTI CONTI SU UN FOGLIO DI BRUTTA, COPIARE SOLAMENTE L'ULTIMA TABELLA DOVE RISULTA CHE LA SOMMATORIA E' ≤ 1 .

Prendere la tabella soprastante ed eseguire diverse **iterazioni** finchè la sommatoria degli U_i non risulta ≤ 1 .

Prendere ogni riga (dalla riga di P_1 alla riga di P_5) e tenerla **bloccata** al fine di calcolare i nuovi T_i **SOTTO** oppure di **SOPRA** che risulteranno (forse) modificati.

NB: i C_i restano gli stessi per tutte le righe!

P_1	T_1	$\uparrow T_1'$
...	...	\uparrow ...
P_k	T_k	$T_k' = T_k$
...	...	\downarrow ...
P_N	T_N	$\downarrow T_N'$

$$\uparrow T_i' = T_{i+1} / \lceil T_{i+1} / T_i \rceil$$

$$\downarrow T_i' = T_{i-1}' * \lfloor T_i / T_{i-1}' \rfloor$$

Se **SALI**
approssimi per
ECESSO
(^parte **BLU**)
Se **SCENDI**
approssimi per
DIFETTO (v parte
VERDE)

Notare che nella formula ogni riga successiva è calcolata usando T_{i-1}' quindi il **risultato ottenuto nella riga precedente!**

Esempio con la tabella di sopra:

1° iterazione \rightarrow Tengo bloccata la riga con P_1 (solo iterazioni per calcolare nuovi T_i verso il basso in quanto sopra P_1 non ho nulla!)

Tengo bloccata la riga di P_1

Calcolo il T_i' di $P_2 \rightarrow 10 * \lfloor 15/10 \rfloor = 10 * 1 = 10$

Calcolo il T_i' di $P_3 \rightarrow 10 * \lfloor 20/10 \rfloor = 10 * 2 = 20$

Calcolo il T_i' di $P_4 \rightarrow 20 * \lfloor 30/20 \rfloor = 20 * 1 = 20$

Calcolo il T_i' di $P_5 \rightarrow 20 * \lfloor 60/20 \rfloor = 20 * 3 = 60$

S'	T_i'	C_i	U_i'
$P_1' = P_1$	10	2	0.2
P_2'	10	4	0.4
P_3'	20	5	0.25

P4'	20	3	0.15
P5'	60	2	0.033

NB: ricorda che ogni **Ui** per ogni riga = **Ci / Ti**.

$$Up = \sum_{i=0}^n \frac{Ci}{Ti} \leq 1$$

Calcolo di Up -> 2/10 + 4/10 + 5/20 + 3/20 + 2/60 = 1.03 > 1 NO -> Ci serve un'altra iterazione

Tengo bloccata la riga di P2. (Ti = 15)

Calcolo il Ti' di P1 -> 15 / ⌈15/10⌉ = 15 * 2 = 7.5 (*)

Calcolo il Ti' di P3 -> 15 * ⌊20/15⌋ = 15 * 1 = 15

Calcolo il Ti' di P4 -> 15 * ⌊30/15⌋ = 15 * 2 = 30

Calcolo il Ti' di P5 -> 30 * ⌊60/30⌋ = 30 * 2 = 60

(*)Se avessi bloccato la riga di P3 e avessi ottenuto Ti' =7.5 in tale riga, salendo verso l'alto avrei sempre riportato il valore 7.5 analogamente a come visto scendendo verso il basso. Essendo però in questo caso bloccata la Riga 2, sopra ho solamente una riga, quindi l'iterazione termina immediatamente usando i valori di P2.

Calcolo di Up -> 2/7.5 + 4/15 + 5/15 + 3/30 + 2/60 = 1 <= 1 **OK** -> **Ci possiamo fermare qui**

S'	Ti'	Ci	Ui'
P1'	7.5	2	0.267
P2' = P2	15	4	0.267
P3'	15	5	0.333
P4'	30	3	0.1
P5'	60	2	0.033

PASSATO, **Quindi crocettare SI/SI.**

Test di Kuo-Mok e corollario di Liu-Layland

Partire dalla tabella INIZIALE del testo della quale si sono calcolati i rispettivi $U_i = C_i/T_i$.

PROCESSO	T_i	C_i	$U_i = C_i/T_i$
P1	10	2	0.2
P2	15	4	0.27
P3	20	5	0.25
P4	30	3	0.1
P5	60	2	0.03

Effettuare dei raggruppamenti tra i processi, in una nuova tabella tale per cui la somma di tutti gli U_i appartenenti a quel gruppo è il più distante possibile dalla somma di tutte le altre U_i appartenenti ad altri gruppi.

NB: I processi che si decidono di unire nello stesso gruppo devono avere il più piccolo T_i del gruppo che è **MULTIPLO DI TUTTI GLI ALTRI T_i** appartenenti allo stesso gruppo.

Esempio: In questo esempio le possibili permutazioni tali per cui il T_i più piccolo (di ogni gruppo) risulta multiplo di tutti gli altri sono 3 (evitando quella in cui avremmo un gruppo singolo che è da evitare).

Relazione armonica: i processi sono di T_i multipli interi fra di loro (l'uno con l'altro, vicendevolmente). Per esempio {10, 20, 60} (10 è multiplo di 20 e 60, anche 20 è multiplo di 60), invece {10, 20, 30} non va bene, siccome 30 non è multiplo di 20.

Il processo P5 di T_i 60 è in relazione armonica con entrambi i gruppi {10, 20} e {15, 30} (che sono "obbligati"). Lo affidiamo al gruppo con maggiore fattore di utilizzazione U_i .

{P1, P3} hanno un U_i' (totale) di $= 0.2 + 0.25 = 0.45$

{P2, P4} hanno un U_i' (totale) di $= 0.27 + 0.1 = 0.37$

P5 può stare sia in {P1, P3}, sia in {P2, P4}, **lo metto nel gruppo con U_i' più alto**, quindi in {P1, P3}.

Risultato e gruppi in relazione armonica: {P1, P3, P5} e {P2, P4}.

La tabella così trovata avrà:

S'	$T_i' [t.u.]$	$C_i' [t.u.]$	U_i'
$P_1' \equiv \{P_1, P_3, P_5\}$	10	4.83	0.483
$P_2' \equiv \{P_2, P_4\}$	15	5.5	0.367

1) **Come U_i'** è la somma dei vari U_i dei processi appartenenti al gruppo

2) **Come T_i'** è il T_i più piccolo del processo in quel gruppo

3) **Come C_i'** si trova facendo la formula inversa ovvero $U_i' * T_i' = C_i' \Rightarrow 0.483 * 10 = 4.83$

Si procede a scrivere la formula richiesta dal prof.

ATTENZIONE => E' possibile avere questo esercizio in 2 differenti varianti:

1) SOLO test di Kuo-Mok (se chiede solo lui, di solito fallisce)

Test Kuo-Mok:

$$U_p \leq U_{RMP O}(N) = N(2^{1/N} - 1), \text{ Se:}$$

- Vero, indicare SI/SI (oppure procedere con Liu-Layland)
- Falso, indicare NO/NO

NB: N=numero di gruppi (di processi). Nel nostro caso N = 2 in quanto abbiamo un gruppo con 3 processi e uno con 2, in totale 2 gruppi.

Valori noti:

- $U_{RMP O}(N = 2) = 0.828$
- $U_{RMP O}(N = 3) = 0.78$
- $U_{RMP O}(N = 4) = 0.757$

2) Test di Kuo-Mok + corollario di Liu-LayLand (eseguire quanto visto al punto 1 e fare anche questo test sottostante)

Corollario del teorema di Liu-Layland:

$$\prod_{i=1}^N (1 + U_i) \leq 2 \text{ Se:}$$

- Vero, indicare SI/SI
- Falso, indicare NO/NO

Nell'esempio:

- Kuo-Mok: $0.45 + 0.37 = 0.82 \leq 2(2^{1/2} - 1) = 0.828 \Rightarrow 0.82 \leq 0.828 \Rightarrow \text{OK}$
- Liu-Layland: $[(1 + 0.45) * (1 + 0.37)] \leq 2 \Rightarrow 1.9865 \leq 2 \Rightarrow \text{OK}$
- Risposta: SI/SI

Test di Burchard

$$X_j = \log_2 T_j - \lfloor \log_2 T_j \rfloor, \quad \forall j$$

$$\zeta = \max_{1 \leq j \leq N} X_j - \min_{1 \leq j \leq N} X_j$$

$$U_{RMPO}(N, \zeta) = \begin{cases} (N-1)(2^{\frac{\zeta}{N-1}} - 1) + 2^{1-\zeta} - 1 & \zeta < 1 - \frac{1}{N} \\ N(2^{\frac{1}{N}} - 1) & \zeta \geq 1 - \frac{1}{N} \end{cases}$$

Test: $U_P \leq U_{RMPO}$ **Se:**

- Vero, indicare SI/SI
- Falso, indicare NO/NO

Esempio con la seguente configurazione di processi (ricorda arrotonda per DIFETTO)

NB: N = numero dei processi quindi = 4 in questo esempio.

	T_i	C_i	U_i
P1	4	1	0.25
P2	6	1	0.17
P3	8	2	0.25
P4	12	2	0.17

$$X_1 = \log_2(4) - \lfloor \log_2(4) \rfloor = 0$$

$$X_2 = \log_2(6) - \lfloor \log_2(6) \rfloor = 0.585$$

$$X_3 = \log_2(8) - \lfloor \log_2(8) \rfloor = 0$$

$$X_4 = \log_2(12) - \lfloor \log_2(12) \rfloor = 0.585$$

$$= 0.585 - 0 = 0.585$$

$$U_P = 0.84 \leq U_{RMPO}(N,) = 0.768 \Rightarrow \text{NO}$$

Risposta: NO/NO

Algoritmo di Audsley

Quando il numero di processi è elevato (come nell'esempio preso in considerazione) è possibile che l'algoritmo di Audsley non venga richiesto.

Per questo esempio prenderemo in considerazione i seguenti processi.

	T_i [t.u.]	C_i [t.u.]
P_1	4	1
P_2	6	1
P_3	8	2
P_4	12	2

Scorrere la tabella VERTICALMENTE da sinistra verso destra, fermarsi nella colonna relativa ad ogni processo se, dopo aver effettuato il calcolo, il numero appena trovato risulta uguale al numero **sopra**. In caso contrario continuare ad applicare la formula per i numeri sottostanti.

1) La Riga **$R_0 = C_i$** quindi basta semplicemente copiare i C_i del testo del problema.

2) Ogni altra riga è calcolata nel seguente modo:

$R_i = R_i$ PRECEDENTE (rimane costante nella sommatoria)

$C_j = C_i$ dei processi di indice MINORE

$T_j = T_i$ dei processi di indice MINORE

Mi fermo appena ne ho 2 uguali per ogni colonna

$$I_i(R_j) = \sum_{j|p_j > p_i} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

$$R_i^0 = C_i$$

$$R_i^n = C_i + I_i(R_i^{n-1}), \quad n = 1, 2, \dots$$

Che in italiano significa semplicemente: (NB approssimazione **PER ECCESSO**)

**$C_{i\text{ProcessoConsiderato}} +$
 **$(\lceil (\text{numeroDiSopra} / T_{i\text{ProcessoConsiderato}} - 1) \rceil *$
 **$C_{i\text{ProcessoProcessoConsiderato-1}} +$
 **$(\lceil (\text{numeroDiSopra} / T_{i\text{ProcessoConsiderato-2}} \rceil *$
 $C_{i\text{ProcessoProcessoConsiderato-2}} + \dots$********

Si tratta di una **sommatoria**, quindi nel caso di P_1 il valore subito sotto sarà pari al valore subito sopra (in quanto non esistono processi più prioritari).

Per P_2 saranno da sommare i valori relativi a P_1 , per P_3 avremo sia P_2 che P_1 mentre per P_4 avremo tutti i processi nella sommatoria.

Vediamo con un esempio:

	P1	P2	P3	P4
$R_i^0 = C_i$	1	1	2	2
$R_i^1 = C_i + I_i(R_i^0)$	a = 1	b = 2	e = 4	g = 6
$R_i^2 = C_i + I_i(R_i^1)$		c = 2	f = 4	h = 7
$R_i^3 = C_i + I_i(R_i^2)$				i = 8
$R_i^4 = C_i + I_i(R_i^3)$				j = 8

a = basta guardare il numero di sopra. L'algoritmo termina sempre per il caso a, in quanto ho 2 numeri uguali nella colonna immediatamente

b è nella colonna di P2 :

Ci proc. considerato Ci di P2

$$b = 1 + ([1/4] * 1) = 1 + 1 = 2$$

Numero di sopra nella stessa colonna

Ti di P1

Ci di P1

Essendo 2, risultato appena trovato, diverso da 1 (numero sopra) devo continuare e fare una successiva iterazione. Attenzione, il numero di sopra della colonna ora diventa pari a 2!

$$c = 1 + ([2/4] * 1) = 1 + 1 = 2 \text{ STOP}$$

e è nella colonna P3 :

Numero sopra stessa colonna

$$e = 2 + ([2/6] * 1) + ([2/4] * 1) = 2 + 1 + 1 = 4$$

Ci proc. considerato Ci di P3

Numero sopra stessa colonna

Ci di P2

Ti di P2

Ti di P1

Ci di P1

Qui notiamo che i processi più prioritari sono 2. Quindi, trovandomi nella colonna di P3 dovrò considerare un pezzo relativo a P2 ed uno relativo a P1.

$$f = 2 + ([4/6] * 1) + ([4/4] * 1) = 2 + 1 + 1 = 4 \text{ STOP}$$

$$g = 2 + ([2/8] * 2) + ([2/6] * 1) + ([2/4] * 1) = 2 + 2 + 1 + 1 = 6$$

$$h = 2 + ([6/8] * 2) + ([6/6] * 1) + ([6/4] * 1) = 2 + 2 + 1 + 2 = 7$$

$$g = 2 + ([7/8] * 2) + ([7/6] * 1) + ([7/4] * 1) = 2 + 2 + 2 + 2 = 8$$

$$h = 2 + ([8/8] * 2) + ([8/6] * 1) + ([8/4] * 1) = 2 + 2 + 2 + 2 = 8 \text{ STOP}$$

Consiglio: Tenersi da parte la tabella con Ti e Ci. Alla fine per i calcoli successivi della stessa colonna basta modificare il dato relativo al numero Sopra. Ti e Ci restano uguali!

La **RISPOSTA** può essere **SÌ/SI** oppure **NO/SI** in base alla situazione ottenuta dopo la tabella: se mi fermo in tutte le colonne SI/SI.

Non strict LST

In pratica: dare priorità di esecuzione ai processi con slack minore. Lo slack mi dice quanto al massimo posso differire l'esecuzione.

Il calcolo dello slack della tabella (e il conseguente valore $t=$ per quella colonna) va rifatto ogni volta che si incontra un periodo (barra nera verticale).

Per questo algoritmo occorrono le Deadline fornite dal testo (se non specificata, la deadline è pari al T_i del processo), nell'esempio considerato abbiamo:

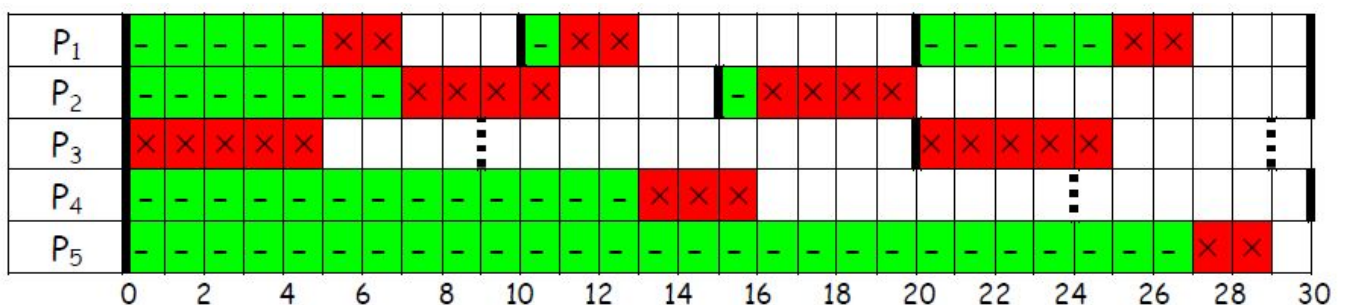
	T_i	C_i	D_i
P1	10	2	10
P2	15	4	15
P3	20	5	9
P4	30	3	24
P5	60	2	40

*Slack = * caselle prima della prossima deadline * - * caselle rosse ancora da eseguire **

Oppure: *Prossima deadline - t + < quante volte ho già eseguito in quel periodo > - C_i*

Il calcolo di una nuova colonna dello slack va fatta ogni volta che si incontra un nuovo periodo (di qualsiasi processo). In questo caso i primi periodi incontrati sono per $t=0$, $t=10$, $t=15$ e infine a $t=20$ (il prof ci dà lo spazio necessario per fare il giusto numero di istanti che gli interessano).

Per quanto riguarda la costruzione del grafico bisogna dare la priorità nell'esecuzione dei processi che possiedono lo SLACK più vicino.



slack	$t=0$	$t=10$	$t=15$	$t=20$
P ₁	8	8	-	8
P ₂	11	4	11	-
P ₃	4	-	-	4
P ₄	21	11	8	-
P ₅	38	28	23	18

	T_i	C_i	D_i
P1	4	1	4
P2	8	3	8
P3	20	4	15
P4	40	2	10

$D_i - C_i$

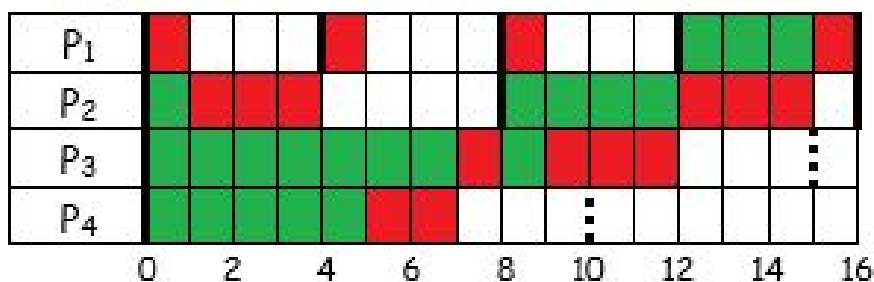
slack	t=0	t= 4	t= 8	t= 12
P ₁	3	3	3	3
P ₂	5	-	5	1
P ₃	11	7	4	-
P ₄	8	4	-	-

A t = 4 ho la prossima deadline tra 4 caselle bianche e devo ancora eseguire 1 in quel periodo $4-1 = 3$

Se ho già eseguito e completato in quel periodo considerato, metti -

A t = 4 ho la prossima deadline tra 6 caselle e devo ancora eseguire 2 in quel periodo $6-2 = 4$

Esegue SEMPRE chi ha lo slack calcolato più corto!



esito positivo

si	no
----	----

risposta conclusiva

si	no
----	----

Da notare che per calcolare il **t=0** la formula è semplicemente **$D_i - C_i$** per il processo considerato.

La costruzione del grafico si effettua dando la priorità al processo con SLACK calcolato minore. Lui esegue, gli altri aspettano.

Il test passa (SI/SI) se TUTTI i processi sono riusciti ad eseguire prima della rispettiva deadline, in caso negativo crocettare NO/NO.

Attenzione alle deadline (per esempio quella di P4 o di P3) e a non confonderle con il periodo: un processo deve eseguire **ENTRO** la deadline, **NON** deve ri-eseguire superata la deadline!

Ricordarsi, se non presenti, di segnare le deadline sul grafico con dei tratteggi!

DMPO

Fattore di utilizzazione efficace del processore

Prendere la tabella originale con l'aggiunta di D_i e δ_i . Alcune D_i sono indicate nel testo, le altre vengono messe pari a T_i . $\delta_i = D_i / T_i$

	T_i	C_i	D_i	δ_i
P1	10	2	10	1
P2	12	1	12	1
P3	15	3	9	0.6
P4	20	6	14	0.7

Ordinare i processi in ordine crescente di D_i (cioè in ordine di priorità) in una nuova tabella

	T_i	C_i	D_i	δ_i
P3	15	3	9	0.6
P1	10	2	10	1
P2	12	1	12	1
P4	20	6	14	0.7

Definire una ulteriore nuova tabella con 4 colonne aggiuntive: H_n , H_1 , f, $U(N, \delta)$

- H_n è il sottoinsieme di processi sopra al processo considerato con $T_i < D_j$
- H_1 è il sottoinsieme di processi sopra al processo considerato con $T_i \geq D_j$

$$f_j = \left(\sum_{i \in H_n} \frac{C_i}{T_i} \right) + \frac{1}{T_j} \left(C_j + \sum_{k \in H_1} C_k \right)$$

$$\sum_{j=1}^N \frac{C_j}{T_j} \leq U(N, \delta) = \begin{cases} N((2\delta)^{1/N} - 1) + 1 - \delta & 0.5 \leq \delta \leq 1 \\ \delta & 0 \leq \delta \leq 0.5 \end{cases}$$

con

$$N = |H_n| + 1$$

$$\delta = \delta_i$$

ovvero la cardinalità di H_n (il numero di elementi) + 1.

Il test risulta passato se

$$f_i \leq U_i(N, \delta) \quad \forall i \in P$$

NB: H_n , H_1 sono sempre vuoti per il primo processo considerato! (P3 qui)

	H_n	H_1	f	$U(N, \delta)$	$f \leq U(N, \delta)$
P3	{}	{}	0.2	0.6	sì
P1	{}	{P3}	0.5	1	sì
P2	{P1}	{P3}	0.53	0.828	sì
P4	{P1, P2}	{P3}	0.73	0.656	no

Per costruire gli insiemi ci si chiede se la Deadline (D) del processo considerato in esame è **maggiore** oppure **minore** di ogni T_i dei processi che si trovano sopra ad esso.

- 1) Se **maggiore o uguale** inserisco il processo più prioritario (quello sopra) in H_n
- 2) Se **minore** inserisco il processo più prioritario (quello sopra) in H_1

P3 => Non esistono processi più prioritari, gli insiemi H_n e H_1 sono entrambi vuoti.

La f vale, sempre e solo per la prima riga, C_i/T_i .

P1 => Confronto D_i di P1(10) con T_i (15) di P3. Risulta $10 < 15$ quindi P3 va in H_1

P2 => Confronto D_i di P2(12) con T_i (15) di P3. Risulta $12 < 15$ quindi P3 va in H_1

Confronto D_i di P2(12) con T_i (10) di P1. Risulta $12 \geq 10$ quindi P1 va in H_n

P4 => Confronto D_i di P4(14) con T_i (12) di P2. Risulta $14 \geq 12$ quindi P2 va in H_n

Se siamo nella prima riga la f è uguale alla U_i del processo della prima riga

$$f(P3) = U_3 = C_i/T_i = 0.2$$

Altrimenti la formula va applicata per intero, considerando che T_j e C_j sono quelli della riga corrente

$$f(P1) = 1/10 * (2 + 3) = 0.5$$

$$f(P2) = 0.2 + 1/12 * (1 + 3) = 0.533$$

$$f(P4) = 0.2 + 0.083 + 1/20 * (6 + 3) = 0.733$$

Nel calcolo della U va considerato il δ_i corrente e come N il numero degli elementi in $H_n + 1$

$$\mathbf{P3} \Rightarrow U(1, 0.6) = (2\delta - 1) + 1 - \delta = \delta = 0.6$$

$$\mathbf{P1} \Rightarrow U(1, 1) = 1$$

$$\mathbf{P2} \Rightarrow U(2, 1) = 2 * ((2 * 1)^{1/2} - 1) + 1 - 1 = 0.828$$

$$\mathbf{P4} \Rightarrow U(3, 0.7) = 3 * ((2 * 0.7)^{1/3} - 1) + 1 - 0.7 = 0.656$$

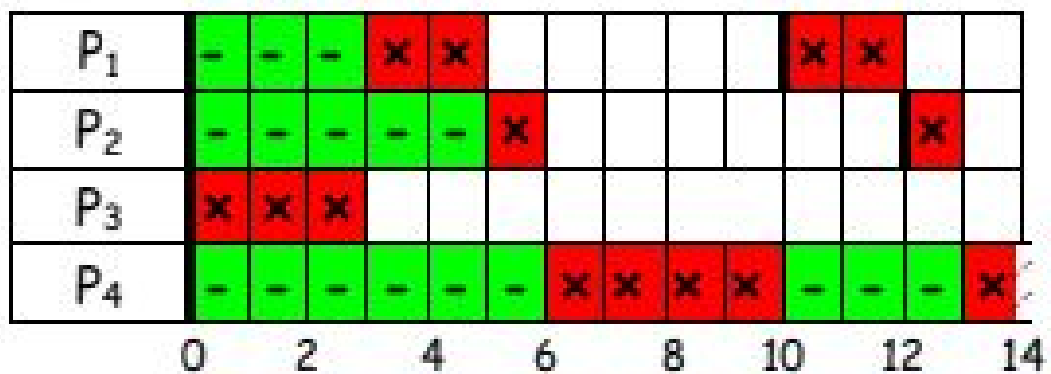
Se la colonna di $f \leq U(N, \delta)$ contiene tutti sì la risposta è **SI/SI**, altrimenti è **NO/NO**

Diagramma temporale

Se non è indicato nel testo, non farlo!

Per costruire il diagramma basta dare priorità di esecuzione al processo con D_i minore (deadline più vicina)

	T_i	C_i	D_i	δ_i
P1	10	2	10	1
P2	12	1	12	1
P3	15	3	9	0.6
P4	20	6	14	0.7



Da notare come P4 dovrebbe eseguire 6 volte e ha come deadline 14. Nel diagramma abbiamo eseguito solo 5 volte, quindi c'è una missed deadline

Se tutte le deadline sono state rispettate scrivere **SI/SI**, altrimenti **NO/SI**

EDF

Test basato sulla densità di utilizzazione del processore

$$\Delta = \sum_{i=1}^N \frac{C_i}{D_i} \leq 1$$

	T_i	C_i	D_i	C_i / D_i
P1	10	2	10	0.2
P2	12	1	12	0.083
P3	15	3	9	0.33
P4	20	6	14	0.428

$$\Delta = 0.2 + 0.083 + 0.33 + 0.428 = 1.041 > 1 \Rightarrow \text{NO}$$

Se $\Delta \leq 1$ la risposta **SI/SI**, altrimenti **NO/NO**

Approccio processor demand

	T_i	C_i	D_i	C_i / D_i	U_i
P1	10	2	10	0.2	0.2
P2	12	1	12	0.083	0.083
P3	15	3	9	0.33	0.2
P4	20	6	14	0.428	0.3

Si comincia calcolando t^*

$$t^* = \frac{\sum_{i=1}^N \left(1 - \frac{D_i}{T_i}\right) C_i}{1 - U_p}$$

con

$$U_p = \sum_{i=1}^N \frac{C_i}{T_i}.$$

$$U_p = 0.2 + 0.083 + 0.2 + 0.3 = 0.783$$

$$t^* = \frac{(1-1)*2 + (1-1)*1 + (1-0.6)*3 + (1-0.7)*6}{1-0.783} = 13.82$$

Proseguire calcolando i BI:

$$BI^n = \sum_{i=1}^N \lceil \frac{BI^{n-1}}{T_i} \rceil C_i$$

con

$$BI^0 = \sum_{i=1}^N C_i.$$

Ci si ferma quando si trovano due BI^n uguali, tale valore è detto BI.

n	BI^n
0	12
1	14
2	15
3	15

$$BI^0 = \sum_{i=1}^N C_i = 2 + 1 + 3 + 6 = 12 \rightarrow$$

$$BI^1 = \lceil \frac{12}{10} \rceil * 2 + \lceil \frac{12}{12} \rceil * 1 + \lceil \frac{12}{15} \rceil * 3 + \lceil \frac{12}{20} \rceil * 6 = 14 \rightarrow$$

$$BI^2 = \lceil \frac{14}{10} \rceil * 2 + \lceil \frac{14}{12} \rceil * 1 + \lceil \frac{14}{15} \rceil * 3 + \lceil \frac{14}{20} \rceil * 6 = 15 \rightarrow$$

$$BI^3 = \lceil \frac{15}{10} \rceil * 2 + \lceil \frac{15}{12} \rceil * 1 + \lceil \frac{15}{15} \rceil * 3 + \lceil \frac{15}{20} \rceil * 6 = 15 \square$$

Si sceglie un valore $m = \min(t^*, BI)$.

$$m = \min(13.82, 15) = 13.82$$

Si calcola l'insieme $D \cap D^*$ che rappresenterà tutti i valori nella tabella alla colonna t .

Formalmente si definisce

$$D \cap D^* = \{d_{ik} \mid d_{ik} = (k - 1) T_i + D_i; d_{ik} < m, 1 \leq i \leq N \wedge k \geq 1; i, k \in \mathbb{N}\}$$

il che significa selezionare per ogni processo tutti quei valori minori di m ottenuti sommando la deadline del processo D_i a tante volte T_i finché ancora verificata tale condizione. In tale formula la k rappresenta semplicemente il valore su cui si cicla finché la condizione non è più verificata, mentre la i il processo preso in considerazione.

E.g. se si ha $T_i = 8$ e $D_i = 7$ e $t^ = 26$, l'insieme $D \cap D^* = \{7, 15, 23\}$ in quanto*

$$D_i = 7, T_i + D_i = 7 + 8 = 15, D_i + T_i + T_i = 23$$

Nell'esempio:

$$D \cap D^* = \{9, 10, 12\} \text{ (in ordine crescente)}$$

A questo punto si può procedere con il riempimento della tabella seguendo la formula:

$$C_p(0, t) = \sum_{i=1}^N C_i(0, t) = \sum_{i=1}^N \left(\left\lfloor \frac{t-D_i}{T_i} \right\rfloor + 1 \right) C_i$$

La tabella è riempita scrivendo in ogni riga il risultato di

$$C_i(0, t) = \left(\left\lfloor \frac{t-D_i}{T_i} \right\rfloor + 1 \right) C_i$$

con t l'elemento corrente dell'insieme $D \cap D^*$

NB: Si ricorda che l'arrotondamento per difetto di un numero negativo, prende il numero negativo intero più piccolo (es. $\lfloor -0.1 \rfloor = -1$)

Nell'esempio:

$$C_1(0, 9) = \left(\left\lfloor \frac{9-10}{10} \right\rfloor + 1 \right) * 2 = (\lfloor -0.1 \rfloor + 1) * 2 = (-1 + 1) * 2 = 0$$

$$C_2(0, 9) = \left(\left\lfloor \frac{9-12}{12} \right\rfloor + 1 \right) * 1 = (\lfloor -0.25 \rfloor + 1) * 1 = (-1 + 1) * 1 = 0$$

$$C_3(0, 9) = \left(\left\lfloor \frac{9-9}{15} \right\rfloor + 1 \right) * 3 = (\lfloor 0 \rfloor + 1) * 3 = 1 * 3 = 3$$

$$C_4(0, 9) = \left(\left\lfloor \frac{9-14}{20} \right\rfloor + 1 \right) * 6 = (\lfloor -0.25 \rfloor + 1) * 6 = (-1 + 1) * 6 = 0$$

$$C_p(0, 9) = C_1(0, 9) + C_2(0, 9) + C_3(0, 9) + C_4(0, 9) = 0 + 0 + 3 + 0 = 3$$

t	$C_1(0, t)$	$C_2(0, t)$	$C_3(0, t)$	$C_4(0, t)$	$C_p(0, t)$	$\leq t$
9	0	0	3	0	3	sì
10	2	0	3	0	5	sì
12	2	1	3	0	6	sì

Se la colonna di $\leq t$ contiene tutti sì la risposta è **SI/SI**, altrimenti è **NO/NO**

PROBLEMA 2

In questo problema occorre gestire delle richieste asincrone (Ra) che vengono eseguite in diversi modi. **NB: SEGNARE IN TUTTI I GRAFICI ANCHE L'ISTANTE IN CUI UNA RICHIESTA ASINCRONA VIENE COMPLETATA (SERVE PER IL SERVER CUS!)**

RMPO

Servizio in background

	a_i [t.u.]	C_i [t.u.]		T_i [t.u.]	C_i [t.u.]
R_{a1}	5	3	P_1	8	2
R_{a2}	15	4	P_2	10	1
R_{a3}	25	2	P_3	12	2
R_{a4}	35	1	P_4	15	2

	P_i	R_a	S
	Idle	None	Idle
-	Ready	-	Ready
...	-	Pending	Waiting
×	Running	Being Served	Running

NB: Ordinare i processi dal T_i più basso al T_i più alto (se non già fatto dal testo) in quanto rappresenta l'ordine di esecuzione dei processi stessi.

Le richieste asincrone sono trattate come processi di priorità minima.

In questo tipo di servizio NON è presente un vero e proprio server:

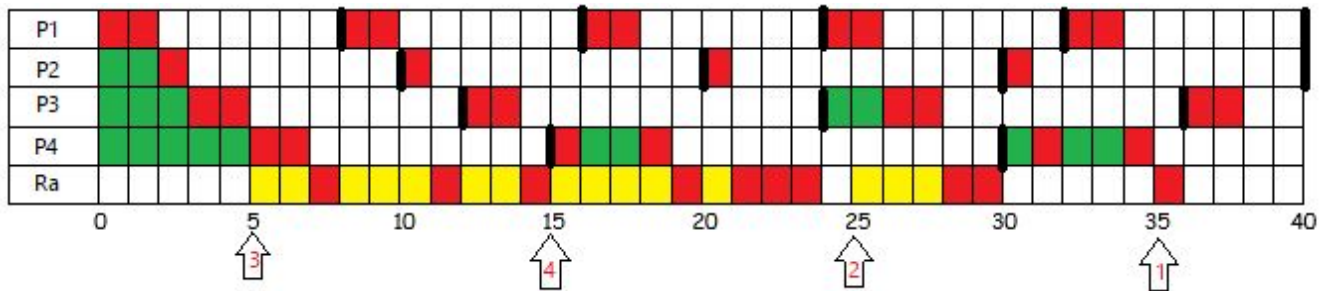
1. Segnare prima in rosso (per ogni periodo) l'esecuzione del processo P_1 in quanto di priorità maggiore.
2. Fare analogamente con tutti gli altri processi all'istante successivo disponibile.
3. Le richieste asincrone sono da gestire come processi di priorità **minima**.

CONSIGLIO: Segnare prima TUTTI gli istanti ROSSI in quanto sono dipendenti dal periodo e dalla priorità.

SIMBOLOGIA:

- **ROSSO** = Per rappresentare l'esecuzione sia di processi che di richieste asincrone.
- **VERDE** = Se il processo vorrebbe eseguire ma non può eseguire perché altri lo stanno già facendo. In sintesi colora tutti gli spazi PRIMA di tutti i rossi in un periodo.
- **GIALLO** = SOLO per le richieste asincrone, rappresenta la volontà di una R_a di eseguire ma l'impossibilità di farlo (attenzione a NON segnalarle se non sono arrivate / non ci sono!)

- **BIANCO** = Quando ho terminato le mie esecuzioni e sono in attesa del prossimo periodo (per un processo) o di una successiva richiesta (per una Ra)



Polling server

In questo caso c'è un server che ha come compito l'esecuzione delle richieste asincrone. Il server ha una sua **capacità**, che indica se il server può o meno eseguire.

- La capacità viene consumata durante il servizio di richieste asincrone
 - Viene fornito dal testo un certo **PERIODO DEL SERVER** T_s e una certa **CAPACITÀ** del server C_s .
 - La capacità del server iniziale viene IMMEDIATAMENTE AZZERATA se non ci sono richieste fino alla prima richiesta.
 - ATTENZIONE: La priorità del server **DIPENDE** dal suo T_s , quindi **si consiglia di riscrivere la Tabella dei processi, inserendo il Server e il suo T_s alla priorità corretta.** Se ordinando i processi ne ho un più prioritario che inizia VEDERE IL CASO PARTICOLARE in quanto lo schema iniziale è diverso!
 - La capacità viene completamente scaricata (**azzerata**), anche se è positiva, quando non sono presenti richieste asincrone.
 - La capacità del server TORNA A LIVELLO MASSIMO ogni suo periodo.
- Segnati quante unità di esecuzione hai eseguito di ogni richiesta per non confonderti

	T_i [t.u.]	C_i [t.u.]
P_1	8	2
P_2	10	1
P_3	12	2
P_4	15	2

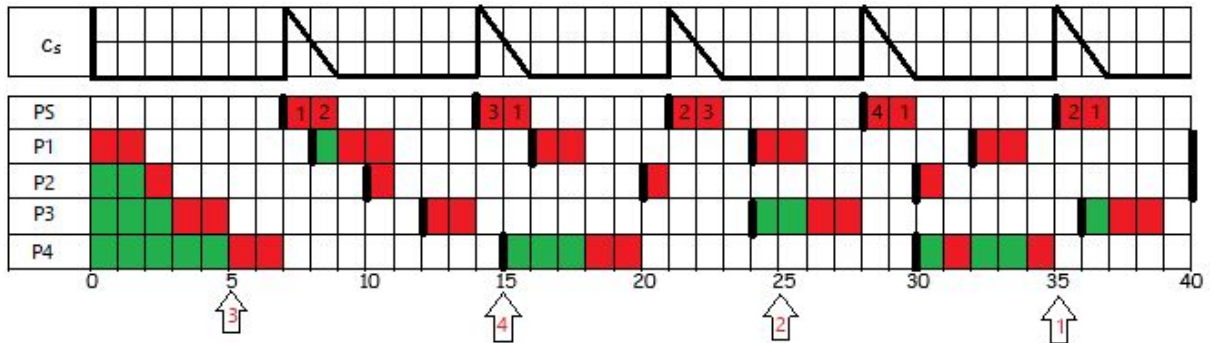
	a_i [t.u.]	C_i [t.u.]
R_{a1}	5	3
R_{a2}	15	4
R_{a3}	25	2
R_{a4}	35	1

$$T_s = 7 \quad C_s = 2$$

NB: La capacità del server nel grafico è alta quanto C_s e parte sempre dal valore massimo
SIMBOLOGIA:

- **ROSSO** = Per rappresentare l'esecuzione sia di processi che di richieste asincrone.
- **VERDE** = Se il processo vorrebbe eseguire ma non può eseguire perché altri lo stanno già facendo. In sintesi colora tutti gli spazi PRIMA di tutti i rossi in un periodo. Vale anche per il server!
- **GIALLO** = Qui non è presente, siccome il server consuma la capacità

- **BIANCO** = Quando ho terminato le mie esecuzioni e sono in attesa del prossimo periodo (per un processo) o di una successiva richiesta (per una Ra)

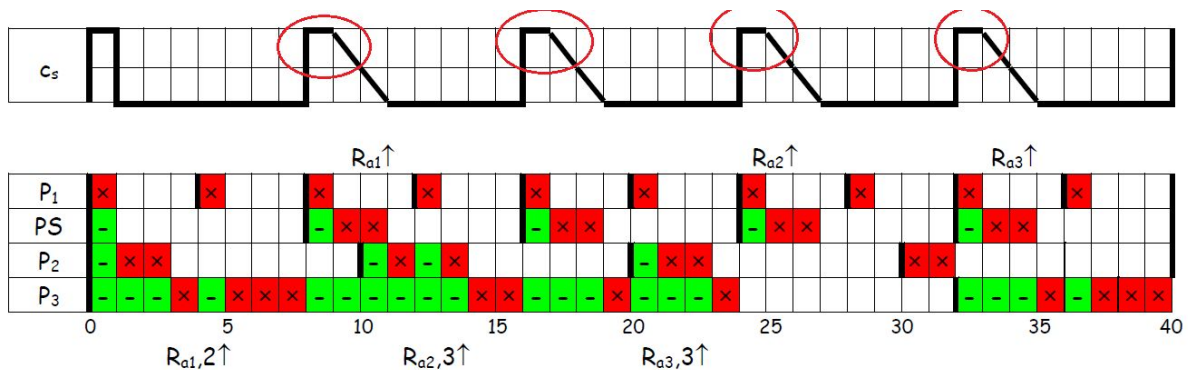


CASO PARTICOLARE:

In questo caso particolare il server NON possiede assoluta priorità in quanto il suo $T_s = 8$ è maggiore di quello di P1 che vale 4.

Essendo P1 quindi più prioritario, occorre **MANTENERE LA CAPACITÀ del server finchè esso non può eseguire.**

Se avessi avuto 2 processi più prioritari del server, avrei mantenuto la capacità anche per 2 unità temporali.



Deferrable server

Come il polling, ma la capacità **si conserva** invece di scaricarsi in assenza di richieste.

Ricordarsi di ricaricare la capacità del server ogni suo periodo.

	T_i [t.u.]	C_i [t.u.]
P_1	10	3
P_2	15	3
P_3	35	7

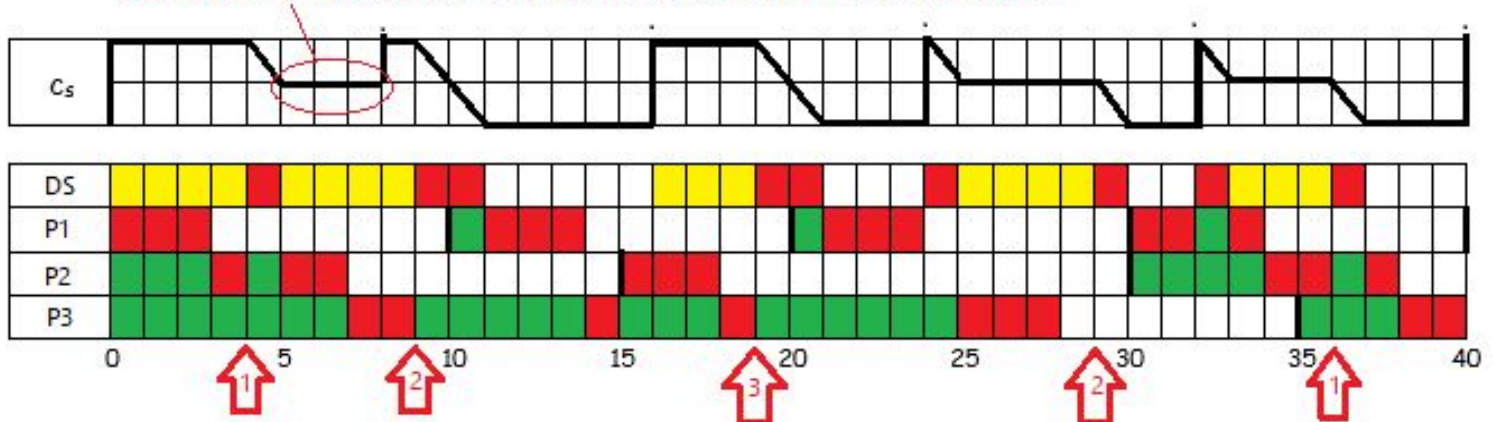
	a_i [t.u.]	C_i [t.u.]
R_{a1}	4	1
R_{a2}	9	2
R_{a3}	19	3
R_{a4}	29	2
R_{a5}	36	1

$$T_s = 8$$

$$C_s = 2$$

Il Deferrable Server va segnato di **GIALLO** quando la sua capacità in qualsiasi punto del grafico risulta MAGGIORE O UGUALE Di 1. Esiste un caso particolare in cui ciò non avviene e invece si colora di **VERDE** ovvero se il server non ha priorità massima (ma non è questo il caso)

Rispetto al polling, la capacità non si azzerava ma si conserva in assenza di richieste!



Priority Exchange Server

In questo tipo di esercizio, la priorità all'esecuzione va data ai processi con il T_i minore. Questo implica che essi debbano essere ordinati in ordine di priorità (server incluso il cui T_s è fornito dal testo).

	T_i [t.u.]	C_i [t.u.]
P_1	10	3
P_2	15	3
P_3	35	7

	a_i [t.u.]	C_i [t.u.]
R_{a1}	4	1
R_{a2}	9	2
R_{a3}	19	3
R_{a4}	29	2
R_{a5}	36	1

NB: LA CAPACITÀ' NON PREGIUDICA L'ESECUZIONE DI PROCESSI, IN QUANTO SERVE SOLAMENTE AL SERVER PER USARLA NELLA GESTIONE DELLE RICHIESTE APERIODICHE

Nel Priority Exchange Server il server ha una sua capacità che inizia al massimo, essa può essere **ceduta** ai processi di priorità inferiore (quelli sotto). Quindi la capacità va sempre verso il basso.

La cessione della capacità può essere:

-Dal **Server a processi di priorità inferiore al server** -> Avviene quando il server **NON ha richieste aperiodiche da gestire** e i processi di sotto (almeno uno) **eseguono**.

-Da **Processi che eseguono a processi che gli stanno SOPRA** (se ho più gente sopra si prende da quello più in alto quindi anche NON adiacenti)

Se tra P_1, P_2 e P_3 **esegue P_2** allora egli può (e lo fa) rubare capacità a un processo che gli sta sopra aumentando la propria capacità da 0 a 1.

Analogamente, all'istante dopo, eseguendo P_3 egli può (e lo fa) rubare capacità ad un processo che gli sta sopra aumentando la propria capacità da 1 a 2.

P_1 $C'=1$ $C=0$ $C=0$

P_2 $C'=0 <$ $C=1$ $C=0$

P_3 $C'=1$ $C=1 <$ $C=2$

$\forall p \in P_{run} C_p = C'_p + 1$ se $\exists p' \in P : p' \gg p \wedge C_{p'} > 0; C_{p'} = C'_{p'} - 1$ altrimenti $C_p = C'_p$

se PES in esecuzione $\wedge C_{pes} = 0 \wedge \exists p : C_p > 0 \Rightarrow C_p = C'_p - 1$

se PES in esecuzione $\wedge C'_{pes} > 0 \Rightarrow C_{pes} = C'_{pes} - 1$

se $\neg \exists p \in P_{run} \exists ! p \in P : C_p > 0 C_p = C'_p - 1$

- La capacità del server, se ce l'ha un processo di priorità inferiore e questo esegue, viene conservata
- Nel caso in cui esistano processi di priorità SUPERIORE al server che eseguono, la capacità del server si mantiene

- In presenza di richieste aperiodiche il server ha la precedenza sui processi inferiori se ha capacità
- La capacità viene riempita al valore massimo ogni periodo T_s
- La priorità del server è pari alla priorità del processo da cui ruba la capacità
- In presenza di idle la capacità più in alto si consuma di 1
- La cessione di capacità avviene solamente verso il basso

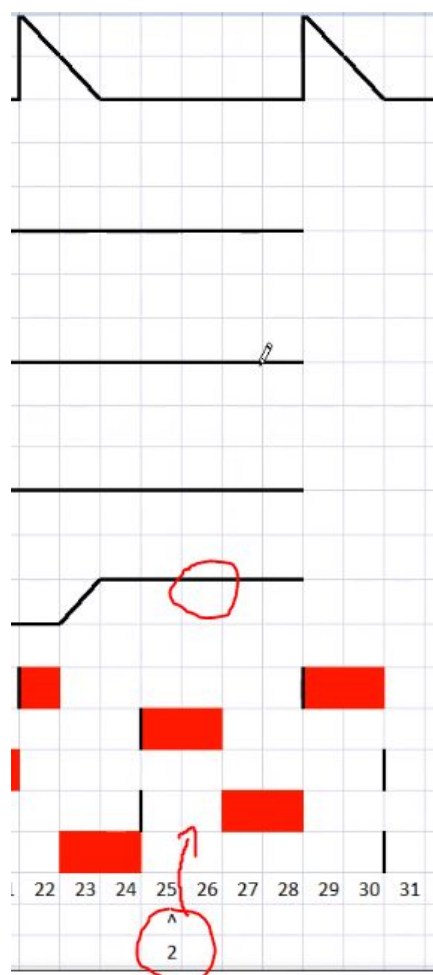
ATTENZIONE SE RUBI AI PROCESSI SOTTO LA CAPACITA' (AS SERVER)!!

Il server, quando ruba capacità, è come se diventasse prioritario come il processo alla quale la ruba!

Quindi, se RUBO capacità a P3 per gestire una richiesta asincrona, posso portare via l'esecuzione solo a P3 o a processi meno prioritari di lui (per esempio P4).

Se Rubassi a P1 (ma non è questo il caso in quanto è scarico) potrei togliere l'esecuzione (e runnare la gestione della richiesta asincrona) a tutti i processi compreso P1.

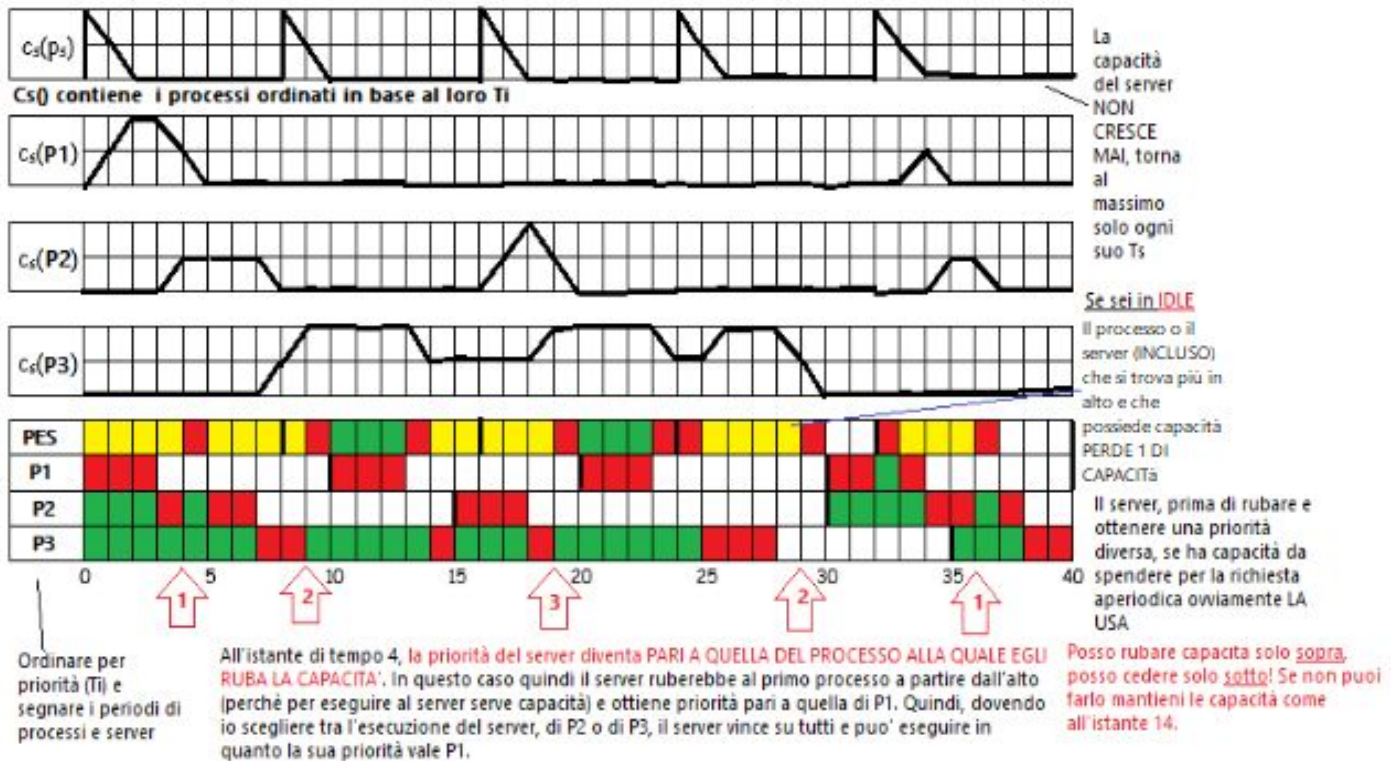
Qui vorrebbe eseguire P1, ruberei da P4 quindi non posso farlo in quanto $P1 > P4$.



Il server è sempre di colore **GIALLO** quando c'è capacità ovunque > 0 .

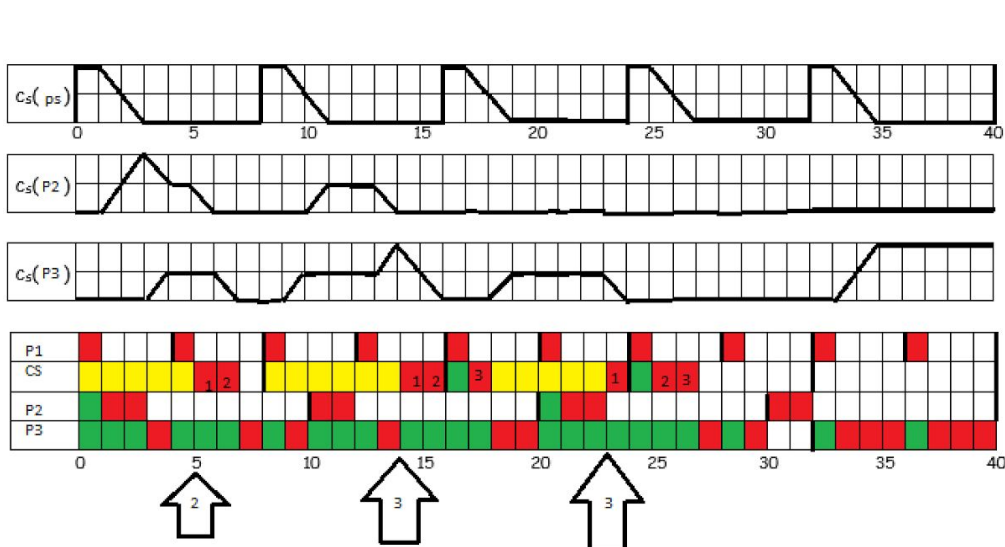
Se la capacità è nulla, il colore è **BIANCO**.

Invece, se il server ha capacità e potrebbe eseguire una richiesta aperiodica, ma è stato bloccato da un processo di priorità superiore, il colore è **VERDE**.



CASO PARTICOLARE: IL SERVER NON E' IL PROCESSO PIÙ PRIORITARIO, IN TAL
 CASI, SE QUALCUNO DI PIÙ PRIORITARIO ESEGUE, LA SUA CAPACITÀ VIENE
 MANTENUTA.

INOLTRE NON OCCORRE DISEGNARE IL GRAFICO DI $cs(P_1)$ in quanto la capacità può
 andare solo verso il basso.



	T_i [t.u.]	C_i [t.u.]
P_1	4	1
P_2	10	2
P_3	32	8

$C_s = 2$
 $T_s = 8$

	a_i [t.u.]	C_i [t.u.]
R_{a1}	5	2
R_{a2}	14	3
R_{a3}	23	3

Sporadic Server :gun:

Questa modalità è piuttosto complicata, quindi partiamo a spiegare da un esempio.

Dati questi processi e queste richieste asincrone:

	T_i [t.u.]	C_i [t.u.]
P_1	10	3
P_2	15	3
P_3	35	7

	a_i [t.u.]	C_i [t.u.]
R_{a1}	4	1
R_{a2}	9	2
R_{a3}	19	3
R_{a4}	29	2
R_{a5}	36	1

Ed il periodo del server preso dal punto 2, è:

2. Server a priorità statica di periodo $T_s = 8$ t.u. e capacità $C_s = 2$ t.u. (strategia di schedulazione dei processi e del Server: RMPO)

Questo esercizio deve essere svolto facendo in parallelo il **grafico colorato**, il **grafico della capacità C_s** e la **tabella**.

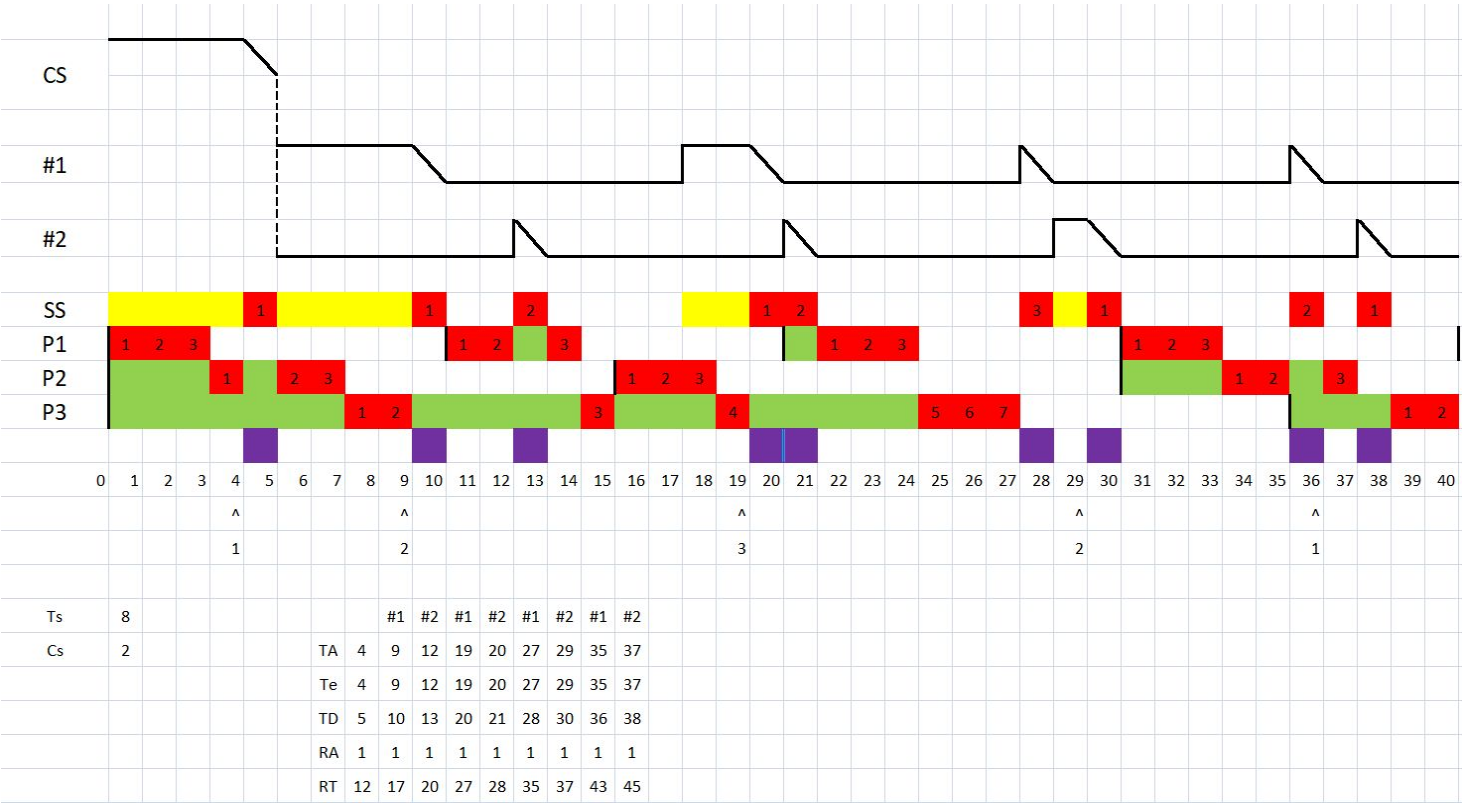
- Il server è **attivo** (Quindi devo segnare un viola sotto) **quando lui o un altro processo di priorità superiore sta eseguendo e la sua capacità è positiva** (vedi esempio 3 con P_1)
- La **divisione** avviene quando non ho consumato tutta la capacità durante la gestione di una richiesta. Le due pistole sono #1 carica (con il valore pari ad 1) ed #2 scarica. Ricordarsi di attivare l'animazione di ricarica SOLO della pistola #2 (la #1 è già carica). La pistola #2 termina la ricarica all'istante RT calcolato nella tabella (Ovvero quando inizia il primo **viola** del blocco + T_s del server).
- La **fusione** avviene se incontro un qualsiasi **viola** (anche causato da processi più prioritari del server che stanno eseguendo) e se entrambe le pistole possono sparare!
- Il server fa iniziare la ricarica quando **INIZIA UN VIOLA, se in quel tratto ho sparato**. Il tempo di ricarica coincide con T_s , quindi dopo T_s si torna full capacità.
- Calcolare un valore della tabella OGNI VOLTA che vedi c'è un nuovo gruppo di **viola**
- t_A è il tempo di inizio del server attivo (quando inizia il viola)
- t_E è primo istante in cui inizio a servire la richiesta aperiodica
- t_D è il tempo di fine del server attivo (quando finisce il viola)
- RA rappresenta quanto la capacità del server è calata durante l'esecuzione della richiesta aperiodica (quanti colpi hai sparato)
- $RT = \max(t_A + T_s, t_D)$
- La capacità viene recuperata dal chunk senza capacità al tempo RT

SIMBOLOGIA:

- **ROSSO** = Per rappresentare l'esecuzione sia di processi che di richieste asincrone
- **VERDE** = Se il processo vorrebbe eseguire ma non può eseguire perché altri di priorità superiore lo stanno già facendo
- **GIALLO** = SOLO per le richieste asincrone, quando il server ha capacità positiva (NB il server può anche essere verde se ho gente + prioritaria che esegue sopra)
- **BIANCO** = Quando ho terminato le mie esecuzioni e sono in attesa del prossimo periodo (per un processo) o di una successiva richiesta (per una Ra). Oppure quando il server ha capacità nulla

Pistolero





EDF

Total Bandwidth Server

Si calcola U_s come il fattore di utilizzazione del server. (U_p = sommatoria dei C_i/T_i)

$$U_s = 1 - U_p$$

Ad ogni istante a_i si ripristina la capacità del server di un valore C_i ovvero il tempo di esecuzione della richiesta attualmente servita.

Devo eseguire per 3 volte? Ti fornisco 3 di capacità!

$$c_s = C_i$$

Si aggiunge nella tabella dei processi una colonna delle deadline d_s calcolata come segue:

$$d_{si} = \max(a_i, d_{s(i-1)}) + \frac{c_s}{U_s}$$

In caso tale valore calcolato sia decimale si considera $d_s' = \lfloor d_s \rfloor$

(ovvero solo la parte intera: se calcolo 11,7 considero come deadline solo 11).

Nel grafico si esegue prima il processo (o la richiesta) con deadline più imminente rivista in ogni istante di tempo.

	T_i	C_i	U_i
P1	4	1	0.25
P2	10	2	0.2
P3	32	8	0.25

$$U_p = 0.25 + 0.2 + 0.25 = 0.7$$

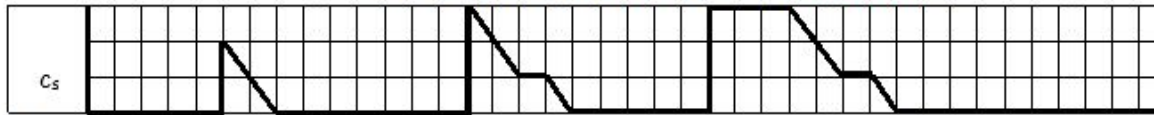
$$U_s = 1 - U_p = 1 - 0.7 = 0.3$$

	a_i	C_i	d_{si}
Ra1	5	2	$5 + (2 / 0.3) = 11.7$
Ra2	14	3	$\max(11.7, 14) = 14 \Rightarrow 14 + (3 / 0.3) = 24$
Ra3	23	3	$\max(23, 24) = 24 \Rightarrow 24 + (3 / 0.3) = 34$

Le deadline sono quindi 11, 24 e 34. OCCORRE SEGNALE SUL GRAFICO, ogni deadline rappresenta la deadline della richiesta i-esima. Quindi Ra1 avrà come scadenza 11, Ra2 come scadenza 14 ed Ra3 come scadenza 23. Tali deadline servono per scoprire chi deve eseguire (priorità dinamica) in quanto, per ogni istante, bisogna valutare chi esegue ovvero chi ha la deadline più vicina (Esegue TBS se è arrivata la richiesta e ovviamente la sua deadline è prima delle altre dei vari processi!).

Attento a non confondere le deadline di Ra1 con quelle di Ra2!!

L'altezza massima è pari al valore **MASSIMO** della richiesta più lunga (quindi pari a 3)
 La capacità **resta sempre a 0** e quando arriva una richiesta **guadagna la capacità** pari alla richiesta ricevuta, se
 eseguo mi scarico. Se vengo interrotto per altri più prioritari **MANTENGO** la capacità



La priorità è dinamica, per convenzione si mette tutto in ordine anche se l'esecuzione è affidata a chi, in ogni istante, ha la deadline più vicina



Il server è SEMPRE **giallo** se il valore di C_s è pari a 0. Se ci sono tratti piani orizzontali, ovvero qualcuno mi ha rubato la priorità, allora è **verde**.

Considerare solo le deadline se il processo ha da eseguire

Deadline delle richieste calcolate con la formula:

$Ra_1 = 11$

$Ra_2 = 24$

$Ra_3 = 34$

Segnati anche quanto durano le Ra e dove iniziano con un tratteggio

La priorità per una Ra in esame è da calcolare in base alla SUA deadline

	T_i	C_i
P1	4	1
P2	10	2
P3	32	8

Se un processo NON ha una deadline esplicita, la sua deadline (da considerare per la priorità dinamica) è pari al suo PROSSIMO periodo quindi $D1 = 4, 8, 12, 16, \dots$ $D2 = 10, 20, 30, 40$ $D3 = 32, 64$. Ovviamente nel calcolo della priorità dinamica occorre considerare il processo in esame per "vincere" la gara di priorità SOLO SE ESSO NON HA ESEGUITO e vuole eseguire. Oppure puoi ignorarlo.

Solitamente nel TBS non vengono date le deadline dei processi, in quanto si assumono come quelle di default i loro T_i ! Per quanto riguarda le deadline delle Richieste asincrone segnare quelle trovate nella tabella superiore con il calcolo!

NOTARE CHE QUI NON CONTA L'ORDINE CON CUI SCRIVO SERVER E PROCESSI NEL GRAFICO, LA PRIORITÀ DIPENDE DALLA DEADLINE!!

Per comodità si scrive sempre prima TBS :)

E' SENSATO SEGNARE LA DEADLINE DI OGNI RICHIESTA (SINGOLARMENTE) nel grafico per non confondersi! (vedi esempio sotto)

Il server è sempre di colore **GIALLO**, se la sua capacità è positiva e non sta eseguendo perché altri processi di priorità superiore eseguono allora colore **VERDE** (Nei tratti piatti del grafico)

Constant Utilization Server (CUS) vs. TBS

PUNTO A)

Il CUS è un esercizio di confronto con il TBS.

1) Riempire la tabella bianca del prof con le stesse deadline per ogni richiesta asincrona calcolate nel punto precedente di TBS:

	a_i [t.u.]	C_i [t.u.]	d_{si} [t.u.]
R_{a1}	6	2	$6 + 2 / 0.346 = 11.8$
R_{a2}	10	1	$11.8 + 1 / 0.346 = 14.7$
R_{a3}	14	4	$14.7 + 4 / 0.346 = 26.2$
R_{a4}	27	2	$27 + 4 / 0.346 = 32.8$

R_{a1} :	$f_1(CUS)$	$f_1(TBS)$
R_{a2} :	$\rightarrow f_2(CUS)$	$f_2(TBS)$
R_{a3} :	$\rightarrow f_3(CUS)$	$f_3(TBS)$
R_{a4} :	$\rightarrow f_4(CUS)$	$f_4(TBS)$

2) Completare i campi delle varie $R_{a1} \dots n$ scrivendo **ai : XXX**

al posto di XXX inserire il rispettivo tempo di arrivo (**ai**) della richiesta, reperibili dal testo d'esame (lasciando la prima R_{a1} in bianco).

3) Indicare dopo la $a_i : xxx$ se tale valore è $>$, $<$ **oppure** $=$ alla deadline i -esima della tabella superiore.

4) In base al segno indicato nella sezione di sinistra delle R_a , abbiamo un corrispondente SEZIONE DESTRA da compilare con il segno corretto da inserire tra $f_i(CUS)$ **XXX** $f_i(TBS)$

I valori a sinistra della R_a indicano che, al posto di XXX occorre inserire:

- R_{a1} segnare sempre $f_1(CUS) = f_1(TBS)$
- $a_i < d_{si}$ allora segnare $f_i(CUS) > f_i(TBS)$
- $a_i > d_{si}$ allora segnare $f_i(CUS) = f_i(TBS)$

Il prossimo passo richiede di completare una tabella che rappresenta il confronto tra i vari server:

Ai fini dell'esempio non si considera **BKG**, **PS** e **PES** ma solamente **TBS** e **CUS** in quanto i valori per i suddetti server sono reperibili dai loro rispettivi grafici.

- a_i e C_i = rispettivamente l'arrivo e la durata di tutte le richieste asincrone.
- f_i = istante in cui ho terminato di servire la richiesta asincrona considerata.
- $f_i - C_i - a_i$ = semplice sottrazione da eseguire.

COLONNA TBS

1) f_i = Quando ho terminato di servire la richiesta asincrona considerata

2) $f_i - a_i - C_i$ = semplice sottrazione tra f_i , il tempo di arrivo della richiesta e il C_i della richiesta.

COLONNA CUS

1) Se f_i (CUS) = f_i (TBS) inserire sia in f_i che in $f_i - a_i - C_i$ un trattino "-"

2) Se f_i (CUS) > f_i (TBS) occorre calcolare il valore con la **formula sotto***

Formula:

$$f_i(CUS) = f_i(TBS) + ds_{i-1} - a_i - < \text{tempo in ready del server nell'intervallo } [a_i; ds_{i-1}] \text{ in TBS} > \quad \forall i$$

Esempio applicazione formula nel caso Ra3:

I due valori da confrontare sono 14 e 14.7, la formula applicata diventa:

$$f_3(CUS) = 21 + 14.7 - 14 - 0 = 21.7$$

Togliere il numero di volte in cui il TBS è ready nell'ultimo pezzo della formula, in questo caso NON è in ready quindi - 0.

$$\begin{array}{ll} R_{a1}: & f_1(CUS) = f_1(TBS) \\ R_{a2}: a_2 = 10 < ds_1 = 11.8 & \rightarrow f_2(CUS) > f_2(TBS) \\ R_{a3}: a_3 = 14 < ds_2 = 14.7 & \rightarrow f_3(CUS) > f_3(TBS) \\ R_{a4}: a_4 = 27 > ds_3 = 26.2 & \rightarrow f_4(CUS) = f_4(TBS) \end{array}$$

	a_i	C_i	f_i	$f_i - a_i - C_i$	f_i	$f_i - a_i - C_i$	f_i	$f_i - a_i - C_i$	f_i	$f_i - a_i - C_i$	f_i	$f_i - a_i - C_i$	f_i	$f_i - a_i - C_i$
R_{a1}	6	2	21	13	12	4	11	3	11	3	8	0		
R_{a2}	10	1	22	11	14	3	12	1	12	1	11	0	12.8	1.8
R_{a3}	14	4	29	11	32	14	21	3	28	10	21	3	21.7	3.7
R_{a4}	27	2	32	3	36	7	29	0	30	1	29	0		
BKG					PS		PES		SS		TBS		CUS	

Vediamo questo **caso particolare**:

	a_i [t.u.]	C_i [t.u.]	d_{si} [t.u.]
R_{a1}	5	2	$5 + 2 / 0.3 = 11.7$
R_{a2}	14	3	$14 + 3 / 0.3 = 24$
R_{a3}	23	3	$24 + 3 / 0.3 = 34$

	a_i	C_i	f_i	$f_i - a_i - C_i$	f_i	$f_i - a_i - C_i$	f_i	$f_i - a_i - C_i$	f_i	$f_i - a_i - C_i$	f_i	$f_i - a_i - C_i$
R_{a1}	5	2	19	12	11	4	7	0	7	0	-	-
R_{a2}	14	3	26	7	26	7	18	1	18	1	-	-
R_{a3}	23	3	30	4	35	9	27	1	30	4	-	-
				BKG	PS		PES		TBS		CUS	

R_{a1} :
 R_{a2} : $a_2 = 14 > d_{s1} = 11.7$ \rightarrow $f_1(CUS) = f_1(TBS)$
 R_{a3} : $a_3 = 23 < d_{s2} = 24$ TBS non è prioritario in [23-24] \rightarrow $f_2(CUS) = f_2(TBS)$
 \rightarrow $f_3(CUS) = f_3(TBS)$

Il server in [23-24] è in ready, quindi applicando la formula di sopra avremmo:

$$f_3(CUS) = 30 + 24 - 23 - 1 = 30 + 1 - 1 = 30$$

Siccome il risultato ottenuto è uguale alla $f_3(TBS)$ possiamo scrivere "nell'intervallo [23-24] TBS non è prioritario"

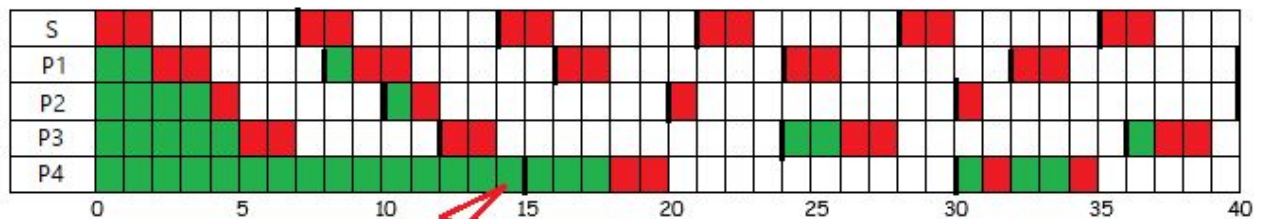
PUNTO B)

a) caso processi P_1, P_2, P_3

Si completa la tabella utilizzando RMPO ricordandosi di elencare i processi (+ il server S) in ordine crescente di T_i .

	T_i [t.u.]	C_i [t.u.]
P_1	8	2
P_2	10	1
P_3	12	2
P_4	15	2

T_i del server = 7
 C_s del server = 2



P4 non riesce ad eseguire entro la sua deadline

La missed deadline avviene se, dopo aver fatto RMPO (senza considerare le richieste asincrone), uno qualsiasi dei processi non riesce a completare la sua esecuzione entro la sua deadline. (Ricorda, se non viene indicata la deadline è uguale al periodo T_i)

Se ho almeno una missed-deadline \Rightarrow Risposta SÌ

Se non ho missed-deadline \Rightarrow Risposta NO

In questo caso la risposta è Sì in quanto ho una missed deadline all'istante 14.

Se ho almeno una missed-deadline $\Rightarrow C_s \leq$ del vecchio valore di C_s

Ora occorre calcolare il C_s , bisogna applicare la formula © Andrea Bisacchi (non il prete)

$$C_s = \frac{< \text{numero caselle rosse processo } S > + < \text{numero tempi in IDLE del sistema} >}{< \text{numero di ripetizioni del server} >}$$

b) caso insieme di processi P_1', P_2', P_3' con stessi U_i e T_i
 (NB U_p = somma di tutti i U_i ovvero C_i/T_i)

$$U_s = C_s / T_s$$

$$U_{s \max} = \frac{2}{(1 + \frac{U_p}{N})^N} - 1,$$

con N = numero di processi tranne il server

Controlliamo che $U_{s \max} \geq U_s$

Se sì, la risposta è NO, viceversa è SI.

Infine, con il T_s del server (testo), calcoliamo

$$C_{s \max} = T_s \cdot U_{s \max}$$

Nell'esempio:

$$U_s = 2 / 7 = 0.286 \quad U_p = 0.65$$

$$U_{s \max} = \frac{2}{(1 + \frac{U_p}{N})^N} - 1 = \frac{2}{(1 + \frac{0.65}{4})^4} - 1 = 0.095$$

$$C_{s \max} = T_s \cdot U_{s \max} = 7 \cdot 0.095 = 0.665$$

$$U_s = 0.286 > U_{s \max} = 0.095 \Rightarrow \text{SI}$$

PROBLEMA 3

Legenda simboli:

- Le **X** si segnano sopra quando è in esecuzione un processo inferiore (cioè di sotto).
- I **O** si segnano sui processi che “vengono saltati” purché siano ready (sfondo verde), partire dal processo che ha ∇ / \square fino a quello che esegue (ovviamente escluso)
- I ∇ / \square si segnano quando cedo la priorità, ovvero quando vengono sospeso (e cedo la priorità)
 ∇ si usa per PCP, IPCP e SRP
 \square si usa per PIP

DMPO

Sono dati un insieme di processi con i rispettivi ϕ_i indicanti gli istanti di tempo di partenza di tali processi:

	T_i [t.u.]	D_i [t.u.]	C_i [t.u.]
P_1	20	20	5
P_2	25	25	6
P_3	35	30	6
P_4	45	45	3
P_5	65	60	6

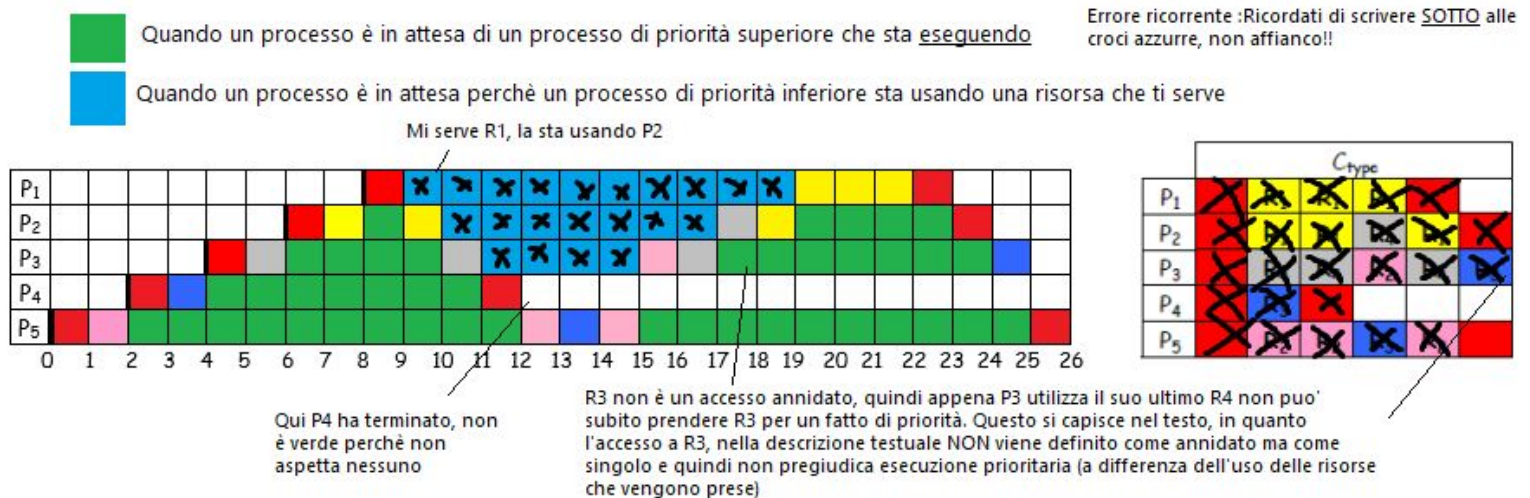
attivati all'istante ϕ_i ([t.u.]) = 8, 6, 4, 2, 0, rispettivamente, è affidata ad un sistema di elaborazione monoprocesso.

Questi processi condividono un certo numero di risorse indicate in questa tabella (NB: controllare che nella descrizione testuale gli accessi siano annidati anche se solitamente è SEMPRE così)

	C_{type}					
P_1		R_1	R_1	R_1		
P_2		R_1	R_1	R_4	R_1	
P_3		R_4	R_4	R_2	R_4	R_3
P_4		R_3				
P_5		R_2	R_2	R_3	R_2	

NOP

POSSO ESEGUIRE? NO PERCHÉ LA RISORSA LA STA USANDO QUALCUN ALTRO, QUINDI NON FACCIO NIENTE E STO FERMO, ESEGUE QUELLO SOTTO DI ME.

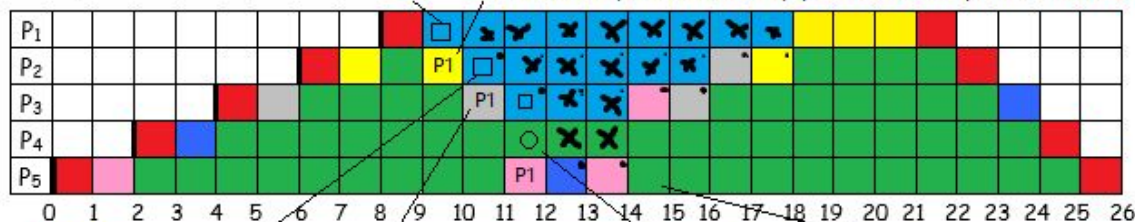


- Comincia a fare eseguire in ordine di precedenza normale con Priorità $P_1 > \text{priorità } P_5$ ma partendo verso il basso in quanto i processi partono in base alla loro ϕ_i , quindi nell'esempio il primo a partire è P5, poi verrà P4, poi P3 etc etc...
- NB: Se un processo inizia con il colore **AZZURRO**, rimane **AZZURRO** finchè non può nuovamente eseguire!!
- Attenzione agli accessi annidati!
Nell'istante **[17-18] P3 non può eseguire usando la risorsa blu R3 in quanto l'accesso annidato è solamente di 4 elementi.**
XX XX
All'istante successivo, P3 non può eseguire **X** in quanto liberando **X** è meno prioritario di P2 che la stava attendendo!

PIP

POSSO ESEGUIRE? NO PERCHÉ LA RISORSA LA STA USANDO QUALCUN ALTRO, QUINDI NON FACCIO NIENTE E STO FERMO, ESEGUE CHI HA LA RISORSA CHE MI SERVIVA PER ESEGUIRE (e fai quadrato perchè cedi la tua priorità a lui).

Se cedo priorità a qualcuno, metto quadrato (1 solo) A P1 serve R1, la sta usando P2, quindi P2 assume priorità P1 finché non libera R1



	Ctype									
P1	X									
P2										
P3										
P4										
P5										

P2 cede la priorità che aveva (P1) a P3, quindi quadrato

A P2 serve R4, la sta usando P3, quindi P3 assume priorità P2 che però aveva assunto priorità P1, quindi la priorità di P3 diventa P1.

A P3 serve R2, la sta usando P5. MA DI MEZZO ABBIAMO P4. Se "scavalco" un processo per ottenere una risorsa che sta usando il processo sotto (anche di 2 posizioni) metti un cerchio al processo che viene scavalcato, in questo caso P4.

Ora che P5 ha terminato di utilizzare R2, P5 perde priorità ereditata da P1. Quindi dovendo scegliere tra l'esecuzione normale di P5 e l'esecuzione di P3 (che detiene ancora priorità P1), P5 attende.

- Un processo che ha ottenuto priorità ereditata da qualcuno, sta **CONTINUANDO** ad avere tale priorità.
- Io, processo che stavo eseguendo, **ho appena ceduto priorità** a qualcuno che sta sotto di me
- Io, processo che stavo in mezzo tra un processo alla quale serviva una risorsa e un altro processo che la deteneva, sono stato scavalcato.
- Io, processo, **ho appena ereditato la priorità segnata all'interno del quadrato** e ho appena eseguito con tale priorità.
NB: Le priorità sono in cascata, se P2 ha priorità P1 ed egli cede priorità a P3, P3 ottiene anch'esso priorità P1

- Se potrei eseguire ma qualcuno mi ha scavalcato perchè è stata ceduta priorità ad un processo sotto di me che non avrebbe potuto eseguire in quanto era meno prioritario di me. In questo caso P4 è stato scavalcato, è per 2 unità di tempo 12-13 e 13-14 potrebbe eseguire, ma P5 che ha ottenuto priorità P1 gli impedisce di farlo
- Quando un processo è in attesa perchè un processo di priorità inferiore sta usando una risorsa che ti serve

Comincia a fare eseguire in ordine di precedenza normale con Priorità P1 > priorità P5 ma partendo verso il basso in quanto i processi partono in base alla loro ϕ_i , quindi nell'esempio il primo a partire è P5, poi verrà P4, poi P3 etc etc...

TUTTI I PROCESSI CHE VENGONO SCAVALCATI DA QUALCUNO CON IL QUADRATO, SONO **VERDI** COL CERCHIO.

Se un processo B ha ereditato priorità da A ed è presente un processo C che eredita da B, allora C prende la priorità di A come nel caso di P3 nell'esempio.

QUI NON CI SONO TRIANGOLI

PCP

POSSO ACQUISIRE UNA RISORSA? LA RISPOSTA ALLA DOMANDA PER OGNI PROCESSO DIPENDE DAL LIVELLO ATTUALE DEL PRIORITY CEILING (IL VALORE). Solo processi di priorità STRETTAMENTE MAGGIORE DEL valore indicato nel PRIORITY CEILING POSSONO ottenere tale risorse, oppure si bloccano (SPESSO CEDENDO IL PROPRIO LIVELLO DI PRIORITÀ AD UNO QUALSIASI DEI PROCESSI IN BASSO ANCHE NON ADIACENTI)

SE RILASCIO UNA RISORSA IL PRIORITY CEILING TORNA AL LIVELLO PRECEDENTE. [Il grafico è considerato solo durante le ACQUISIZIONI delle risorse, durante esecuzioni normali se un processo può eseguire, esegue.]

I processi eseguono in esecuzione normale considerando le loro ϕ_i

Si comincia analizzando la tabella delle risorse, segnandosi il processo che usa la risorsa i-esima in base alla priorità del processo PIÙ ALTO che la sta usando.

Ad esempio R1 è usata da P1 e P2 quindi il processo massimo che la usa ha priorità P1.

	C_{type}					
P_1		R_1	R_1	R_1		
P_2		R_1	R_1	R_4	R_1	
P_3		R_4	R_4	R_2	R_4	R_3
P_4		R_3				
P_5		R_2	R_2	R_3	R_2	

Processi più prioritari per ogni risorsa:

R1 -> P1

R2 -> P3

R3 -> P3

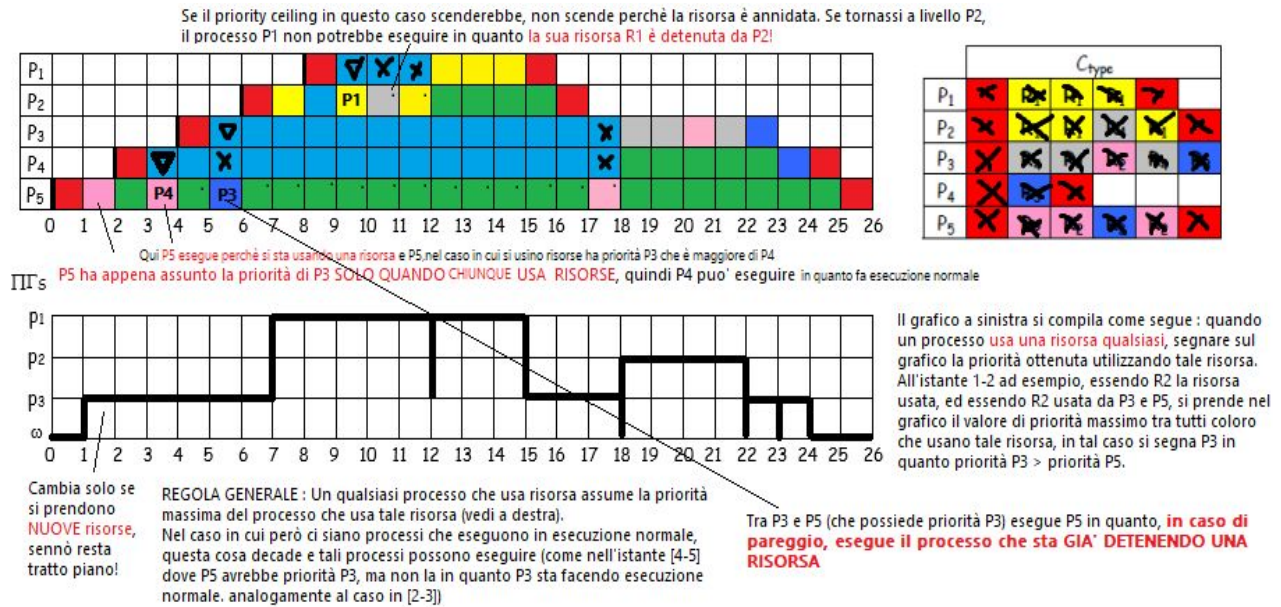
R4 -> P2

- Ogni volta che
 - 1) Una NUOVA risorsa viene presa da un processo viene modificato il priority ceiling al valore corrispondente calcolato sopra.
 - 2) Una risorsa viene rilasciata da un processo: il priority ceiling ritorna al valore precedentemente imposto o a zero.
- Quando un processo stabilisce un priority ceiling acquisendo una nuova risorsa **NON può auto-bloccarsi**, egli continuerà a correre a meno che un processo non batta il priority ceiling che ha appena impostato.
- Se un processo, impostando il priority ceiling, blocca dei processi soprastanti, il processo corrente **EREDITA LA PRIORITÀ DEL PROCESSO BLOCCATO**.

SIMBOLOGIA: Valgono le regole degli altri esercizi, con l'aggiunta dei triangoli.

Inserire i triangolini con la punta verso il basso solo una volta (seguito da tante croci verdi finchè sono bloccato) nei momenti in cui il processo considerato NON ha potuto eseguire a

causa di un qualsiasi processo che si trova sotto di lui, che ha ottenuto potere prioritario



rispetto al processo considerato.

ES: Se sono P4 e batterei P5 durante la normalità, ma P5 ha ottenuto priorità P1,P2 oppure P3 per qualche motivo, allora metto il triangolino su P4 perché non può eseguire e continuo a crocettare finché P5 non ritorna a priorità inferiore alla mia (dopo il rilascio della risorsa)

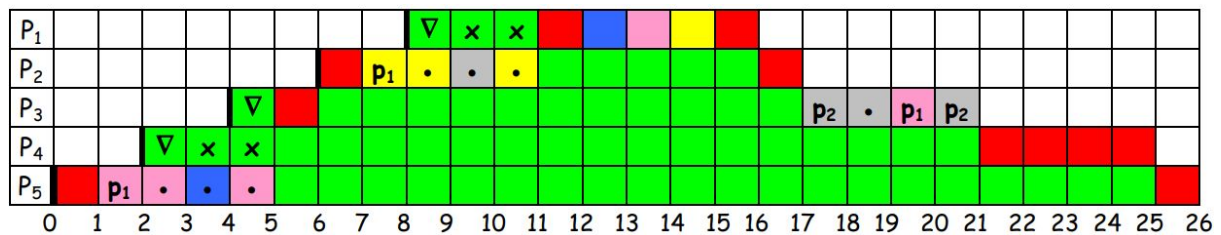
IPCP

Uguale a PCP, unica differenza:

- Non appena un processo acquisisce una risorsa, questo assume priorità pari a quella del processo di priorità massima che utilizzerebbe tale risorsa (anche se il processo di priorità superiore non è partito o ha terminato).
- Nel grafico non ci sono **BLU**, solo **VERDI**.

	T_i [t.u.]	D_i [t.u.]	C_i [t.u.]
P_1	20	20	5
P_2	25	25	6
P_3	35	30	5
P_4	45	45	4
P_5	50	50	6

	C_{type}					
P_1		R_3	R_2	R_1		
P_2		R_1	R_1	R_4	R_1	
P_3		R_4	R_4	R_2	R_4	
P_4						
P_5		R_2	R_2	R_3	R_2	



Massimo tempo di blocco

- PIP $\rightarrow B_i = \min(\sum \max(\text{ogni riga sotto}), \sum \max(\text{ogni colonna a partire dalla riga sotto}))$
- PCP/IPCP $\rightarrow B_i = \max(\text{processi sottostanti } [= \text{ righe sotto}])$

Tabella da completare verticalmente:

- Quante volte P1 usa R1? 0 quindi lascio vuoto
- Quante volte P1 usa R2? 1
- Quante volte P3 usa R2? 3 perché R2R2R2 è accesso annidato.
- Quante volte P3 usa R1? 6 perchè devi considerare gli accessi annidati ad R1.

Una volta fatta la tabella, occorre calcolare i B_i .

	C_{type}														
P ₁			R ₂	R ₄											
P ₂		R ₁													
P ₃	R ₁	R ₄	R ₄	R ₄	R ₄	R ₁		R ₂	R ₄	R ₂		R ₃	R ₄	R ₄	R ₃
P ₄															
P ₅		R ₃	R ₃												

Si prende la tabella e si calcolano i B_i come segue:

	R ₁	R ₂	R ₃	R ₄	B_i	
P ₁		1		1	4	4
P ₂	1				6	6
P ₃	6	3	4	4	2	2
P ₄					2	2
P ₅			2		0	0
	PIP				PCP	

N.B.: ricordarsi che vogliamo il massimo tempo di blocco, cioè il caso peggiore

Colonna **PIP**

1. Guardare la prima tabella con le esecuzioni normali/risorse (qui contano gli accessi annidati).
P1 usa solo R2 e R4, controllare sotto tutti i processi che usano tali risorse (ne basta una). Entrambe sono usate da P3. R4 viene usata 4 volte al massimo, R2 viene usata 3 volte (accesso annidato). Basta prendere il valore massimo, cioè 4.
2. P2 usa solo R1, che è usata ancora da P3. Siccome l'accesso è annidato, P3 usa 6 volte R1.
3. P4 non usa risorse, ma qui basta guardare P5 che nel caso peggiore lo bloccherà per 2 istanti di tempo (accesso a P5)

Colonna **PCP**

1. Guardare la tabella affianco ai Bi costruita precedentemente.
Iniziamo con P1 NON considerando la sua riga (cancellandola mentalmente) ai fini del calcolo del valore Bi della tabella.

Si considerano **solamente tutte le colonne sottostanti** in cui P1 usa almeno 1 risorsa (in questo caso solo R2 ed R4 in quanto gli altri spazi sono vuoti).

Il valore da segnare in tabella è semplicemente il **MASSIMO** tra tutti i numeri presenti nelle colonne (verticalmente) "sbloccate" dal processo. ($\text{Max}(1,3,1,4) = 4$)

	R ₁	R ₂	R ₃	R ₄
P ₁		1		1
P ₂	1			
P ₃	6	3	4	4
P ₄				
P ₅			2	

2. L'arrivo di un nuovo processo può sbloccare ulteriori colonne nella considerazione del massimo. P2, oltre a far cancellare mentalmente la sua riga e di tutti quelli sopra (P1), prende in considerazione la risorsa R1 in quanto la utilizza. A questo punto per il calcolo del massimo avremo: $\text{Max}(1,6,1,3,1,4) = 6$ in quanto la colonna di R1 è stata sbloccata.

	R ₁	R ₂	R ₃	R ₄
P ₁		1		1
P ₂	1			
P ₃	6	3	4	4
P ₄				
P ₅			2	

3. L'arrivo di P3 sblocca **tutte le restanti colonne** in quanto utilizza tutte le risorse, viene cancellata la sua riga e quelle soprastanti e si resta semplicemente con la riga bianca di P4 e un 2 (della riga di P5), il risultato è quindi solo 2.

	R ₁	R ₂	R ₃	R ₄
P ₁		1		1
P ₂	1			
P ₃	6	3	4	4
P ₄				
P ₅			2	

4. Analogamente per P4 avremo lo stesso risultato, mentre per P5, non avendo numeri, scriveremo 0.

CASO PARTICOLARE ESAME DIFFERENTE COLONNA PIP:

La colonna di **PIP** è tuttora oggetto di discussione. Alcune volte, applicando il metodo classico della cancellazione delle righe presente nel formulario le cose tornano:

	C _{type}				
P ₁		R ₁	R ₂	R ₂	R ₁
P ₂		R ₁	R ₁		
P ₃		R ₂	R ₃	R ₂	
P ₄		R ₃	R ₄	R ₃	
P ₅		R ₄	R ₄	R ₄	

	R ₁	R ₂	R ₃	R ₄
P ₁	4	2		
P ₂	2			
P ₃		3	1	
P ₄			3	1
P ₅				3

1. Cancellare la riga di P1 (tabella di destra) e fare la somma di tutti i valori **massimi** presenti in ogni riga: $2 + 3 + 3 + 3 = 11$
2. Cancellare la riga di P2 (e quindi anche di P1) e fare la somma di tutti i valori **massimi** presenti in ogni riga: $3 + 3 + 3 = 9$

Analogamente per gli altri P3 ($3+3 = 6$), P4 (3) e infine P5 (0).

Audsley (the return)

Seguire quanto visto nell'algoritmo di Audsley del PROBLEMA 1 considerando però come unica differenza il calcolo della R0 iniziale.

$$R_i^0 = C_i + \max\{B_{iPCP}, B_{iPIP}\}$$

Il valore della R0 per ogni processo è calcolato con la formula sopra. Nel nostro esempio e, considerando i BI sia di PCP che di PIP calcolati nel punto precedente avremo:

	T_i [t.u.]	D_i [t.u.]	C_i [t.u.]
P_1	20	20	5
P_2	25	25	6
P_3	35	30	6
P_4	45	45	3
P_5	65	60	6

$$R_1^0 = 5 + \max(13, 4) = 18$$

$$R_2^0 = 6 + \max(9, 4) = 15$$

$$R_3^0 = 6 + \max(5, 4) = 11$$

$$R_4^0 = 3 + \max(4, 4) = 7$$

$$R_5^0 = 6 + \max(0, 0) = 6$$

Si procede con [Audsley](#) nel modo classico.

Verificare alla fine se la R_i di Audsley è \leq al BI del processo.

Se c'è almeno un no la risposta è sempre conclusiva **NN/SS**

EDF

SRP

	T_i [t.u.]	D_i [t.u.]	C_i [t.u.]
P_1	15	15	3
P_2	20	20	3
P_3	25	23	3
P_4	30	26	6
P_5	35	29	6

ϕ_i ([t.u.]) = 7, 5, 4, 2, 0,

	C_{type}					
P_1		R_1				
P_2		R_2				
P_3						
P_4		R_1	R_1	R_2	R_2	R_1
P_5		R_2	R_2	R_2	R_2	

unità disponibili per ogni tipologia di risorsa sia rispettivamente $u_1 = 2$, $u_2 = 3$,

	R_1	R_2
P_1	2	
P_2		1
P_3		
P_4	1	2
P_5		2

Le esecuzioni di qualsiasi processo o l'acquisizione di nuove risorse può avvenire SOLAMENTE se il livello di priorità del processo è STRETTAMENTE MAGGIORE al livello di priorità del grafico sottostante. in tutti gli altri casi mi sospendo

- Riempire una nuova tabella ponendosi questa domanda: **“Chi è il processo di priorità maggiore che non può eseguire se ho disponibili $u = x$ unità?”**
 - Se, per esempio $u = 1$ per la colonna R2 è P4, scrivere pl4 in quanto processo massimo che non può eseguire (P4 userebbe 2 unità, e ne abbiamo solo 1).
 - Se tutti possono eseguire, scrivere ω (che indica la priorità minima del sistema, più bassa anche di quella dell'ultimo processo)
- La tabella indicherà per ogni unità disponibile di una risorsa, il tetto di priorità

Disegnare il grafico:

- Se una qualsiasi risorsa acquisita farebbe SCENDERE il lv di priorità, questo non succede MAI in quanto il tetto di priorità scende solamente durante un RILASCIO. Anche in questo caso si prende quindi il MASSIMO a parità di acquisizioni.
- Ogni volta che si acquisisce una risorsa, cambia il tetto di priorità.
Se P5 utilizza 2 unità di R2 e ce ne sono disponibili 3, si deve guardare la colonna $u = 1$ (rimane solo un'unità disponibile ora) della tabella dei livelli di priorità per sapere come cresce il grafico.
- Anche qui a parità di priorità, esegue il processo che detiene la risorsa ed è in esecuzione
- Al rilascio di una risorsa, cambia il livello di priorità nel grafico plx che torna al lv prec.

Prima cosa da fare, riempire la tabella dei plx

unità disponibili per ogni tipologia di risorsa sia rispettivamente $u_1 = 2, u_2 = 3$,

	R ₁	R ₂
P ₁	2	
P ₂		1
P ₃		
P ₄	1	2
P ₅		2

	$u = 0$	$u = 1$	$u = 2$	$u = 3$
R ₁	pl1	pl1	ω	—
R ₂	pl2	pl4	ω	ω

Non ci possono essere 3 unità libere per R1 in quanto da testo ne abbiamo solo 2

Metti ω dove non ci sono processi bloccati (quindi se tutti possono runnare) con tot unità disponibili per quella risorsa

Se ci sono 0 unità libere di R1 chi è il processo di priorità più elevata che NON può eseguire?

Se ho 1 unità libera di R1 chi è il processo di priorità più elevata che NON può eseguire? P1, perchè ne usa 2

il valore del grafico sotto cambia nel seguente modo:

quando P5 acquisisce ad esempio R2 all'istante [1-2] occorre controllare la tabella VERDE

Tale tabella mi dice QUANTE RISORSE ogni processo prende di ogni tipo.

Nel caso di P5 che prende una R2 possiamo leggere nella seconda colonna che ne prende 2 unità.

Controllando il testo sappiamo che il numero di unità disponibili di $u_2 = 3$, quindi si fa la sottrazione

$u_2 - \text{acq} = 3 - 2 = 1$

il valore 1 indica la COLONNA da guardare nella tabella dei plx. Quindi guardando la colonna con $u = 1$ e incrociandola con la risorsa R2, ottengo come nuovo valore per il grafico pl4

Da questo momento P5 ottiene priorità pl4 pari quindi a quella di P4. All'istante successivo, se devo scegliere tra i due si privilegia (come sempre) l'esecuzione del processo che sta usando e all'istante successivo usa un'altra risorsa (in caso di pareggio).

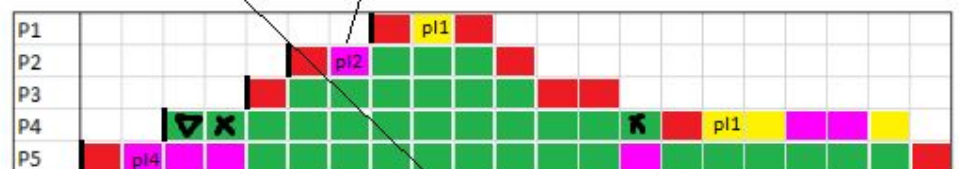
Il grafico dei plx cambia ogni volta che si acquisisce una NUOVA risorsa

Il grafico del px cambia anche quando si RILASCIA una risorsa, si ritorna al livello di priorità precedente (se presente) o nullo

In questo punto P5 non ha ancora rilasciato la risorsa R2 e avendo $u_2 = 3$ con P5 che si è preso 2 unità su 3 ne abbiamo ancora 1 disponibile.

Per tale motivo, P1 è in grado di prendersi l'ultima unità disponibile di R2 in quanto, nella tabella VERDE ne richiede solo 1 unità.

il valore di unità disponibili diventa quindi $u_2 = 0$ che, nella tabella plx corrisponde alla prima colonna facendo portare il lv. di priorità a pl2.



Ricorda che negli accessi annidati il livello di plx del grafico non cambia e il rilascio avviene solo alla fine dell'uso delle risorse!!

	C _{Type}
P ₁	
P ₂	
P ₃	
P ₄	
P ₅	

NB: In questo esercizio compare il simbolo del ∇ . Esso è da inserire ogni volta che un processo **NON** ha potuto eseguire a causa del fatto che la sua priorità era **INFERIORE** al livello di priorità del grafico. Come accade per P4 nell'esempio di sopra.

Questa tabella è la stessa di TBS

NB: L'ultima colonna è il numero di verdi per ogni processo.

f_i rappresenta ogni istante in cui un processo termina con la sua ultima cosa da fare

Ci sono quelli del testo

ϕ_i viene data dal testo, è lo sfasamento di partenza (ritardo) per ogni processo.

	ϕ_i	C_i	f_i	$f_i - \phi_i - C_i$
P_1	7	3	10	0
P_2	5	3	11	3
P_3	4	3	13	6
P_4	2	6	20	12
P_5	0	6	21	15
SRP				

C'è sempre la solita tabella con i f_i .

Calcolo del massimo tempo di blocco [NON FATTO NEL 2020]

Qui occorre calcolare i B_i che però sono spesso gli stessi di PCP/IPCP:

$$B_i = \max\{Z_{jk} : pl_j < pl_i \wedge pl_j < \Pi\Lambda_k(0)\} \quad \forall i$$

A parole: il massimo tempo di utilizzo di una risorsa (Z_{jk}) purché: tale valore sia riferito a un P_j di priorità inferiore al processo corrente (P_i), il livello di priorità di P_j sia inferiore del livello di priorità ($\Pi\Lambda(u_k)$) acquisibile per tale risorsa k assumendo che ce ne siano $u_k = 0$ libere.

In poche parole: dalla tabella di sinistra si eliminano (per il calcolo) le prime caselle con un valore per ogni risorsa. Inoltre elimini le righe sopra la riga corrente, esempio: se calcoli la B_i di P_3 elimino le righe sopra a P_3 (P_3 compreso).

	R ₁	R ₂	B _i		
P ₁	1		9	5	5
P ₂		1	9	5	5
P ₃			9	5	5
P ₄	5	2	4	4	4
P ₅		4	0	0	0
			PIP	PCP	IPCP

	R ₁	R ₂	B _i
P ₁	1		5
P ₂		1	5
P ₃			5
P ₄	5	2	4
P ₅		4	0

La tabella SOPRA è di PCP/IPCP, questa a sinistra è invece **quella che pare essere la sua copia**. La formula di Fabri è utile ma non include casi visti finora in qualsiasi esame. Per ora la copia resta la cosa più veloce **DA VERIFICARE SE CI SIANO ESERCIZI DOVE LE 2 TABELLE NON SONO UGUALI**.

Analisi di schedulabilità [NON FATTO NEL 2020]

$$\forall i \in P, \sum_{k=1}^i \left(\frac{C_k}{D_k} \right) + \frac{B_i}{D_i} \leq 1$$

	R₁	R₂	B_i
P ₁	1		5
P ₂		1	5
P ₃			5
P ₄	5	2	4
P ₅		4	0

	T_i [t.u.]	D_i [t.u.]	C_i [t.u.]
P ₁	15	15	3
P ₂	20	20	3
P ₃	25	23	3
P ₄	30	26	6
P ₅	35	29	6

NB: I valori per ogni processo vanno da 1

fino al processo considerato. Per P4 ad esempio ci sono i termini di P1, P2, P3 e P4

P1: $C_1 / D_1 + B_1 / D_1 = 3 / 15 + 5 / 15 = 0.53 \leq 1$

P2: $C_1 / D_1 + C_2 / D_2 + B_2 / D_2 = 3 / 15 + 3 / 20 + 5 / 20 = 0.8 \leq 1$

P3: $C_1 / D_1 + C_2 / D_2 + C_3 / D_3 + B_3 / D_3 = 3 / 15 + 3 / 20 + 3 / 23 + 5 / 23 = 0.7 \leq 1$

P4:

$$C_1 / D_1 + C_2 / D_2 + C_3 / D_3 + C_4 / D_4 + B_4 / D_4 = 3 / 15 + 3 / 20 + 3 / 23 + 6 / 26 + 4 / 26 = 0.865 \leq 1$$

P5:

$$C_1 / D_1 + C_2 / D_2 + C_3 / D_3 + C_4 / D_4 + C_5 / D_5 + B_5 / D_5 = 0.918 \leq 1$$

NB: $B_5 / D_5 = 0 / 29 = 0$

In questo caso sono tutti minori o uguali di 1 quindi **SS**

NB: basta un solo No per far fallire il test con **NN** in caso contrario inserire **SS**.

***Nota sulle risorse annidate generali:**

In qualsiasi esercizio viene spesso chiesto di aggiornare grafici durante l'acquisizione/rilascio di risorse.

In presenza di **Risorse annidate** fa cambiare le cose:

ES risorsa annidata singola

P2 -> **R1 R2 R1**

In questo caso, un processo più prioritario **P1** dovrà

1) Se vuole **R1 ESTERNA deve aspettare PER FORZA LA FINE DI TUTTE le operazioni di P2**

2) Se vuole la **R2 INTERNA può attendere fino all'istante 2, dopodichè può subentrare e non attendere l'esecuzione della R1 in quanto più prioritario.**

Il lock di una risorsa arriva dal PRIMO uso della risorsa stessa!

Nel caso di risorse annidate ci sono SEMPRE **due acquisizioni e due rilasci** per le risorse coinvolte.

P2 acquisisce R1 poi acquisisce R2, rilascia R2 e infine rilascia R1.

Per questo motivo P1 può prendersi la R2 subito dopo il rilascio di P2!

Le risorse annidate si riconoscono perchè una risorsa X viene ri-utilizzata dopo una certa risorsa Y.

Possono essere anche multiple come nel caso **R1 R1 R2 R2 R1**

Attenzione a non confondersi con il TETTO di priorità, perché in alcuni grafici si deve prendere il MASSIMO derivante dall'operazione annidata, ed è per questo che il grafico non viene aggiornato in quei casi!

Se per esempio in un esercizio che chiede di fare il priority level abbiamo

R1 -> priorità 2

R2 -> priorità 3

Il grafico NON si aggiornerebbe ma resterebbe a priorità 2 in quanto la priorità di R2 è inferiore a quella di R1!